Osnovni principi logičke sinteze (prvi deo)

Stefan Nikolić

Departman za matematiku i informatiku
Prirodno-matematički fakultet, Novi Sad

23.10.2024.

Koraci u generisanju lejauta

Reprezentacije Bulovih funkcija

Bulova funkcija

Preslikavanje $f : \mathbb{B}^n \to \mathbb{B}, \qquad \mathbb{B} = \{0, 1\}$

Istinitosne tablice (eng. Truth Tables)

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

Istinitosne tablice (eng. Truth Tables)

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

Istinitosne tablice su kanonske

(f = g akko tt(f) = tt(g))

Kako da uporedimo dve istinitosne tablice u C-u?

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

Istinitosne tablice su kanonske

(f = g akko tt(f) = tt(g))

Naivno rešenje

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

```
bool ttf[8] =  {1, 1, 0, 0, 0, 1, 1, 1};
bool ttg[8] = {1, 1, 0, 0, 0, 1, 0, 1};

bool eq = 1;
for(unsigned i = 0; i < 8; ++i)
{
   if(ttf[i] != ttg[i])
   {
      eq = 0;
      break;
    }
}
```

Kako to da poboljšamo?

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

```
bool ttf[8] =  {1, 1, 0, 0, 0, 1, 1, 1};
bool ttg[8] = {1, 1, 0, 0, 0, 1, 0, 1};

bool eq = 1;
for(unsigned i = 0; i < 8; ++i)
{
   if(ttf[i] != ttg[i])
   {
      eq = 0;
      break;
    }
}
```

Operacije nad bitovima

f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

```
unsigned ttf = 0xE3;
unsigned ttg = 0xE1;

bool eq = ttf == ttg;
```

Za funkcije do 6 promenljivih

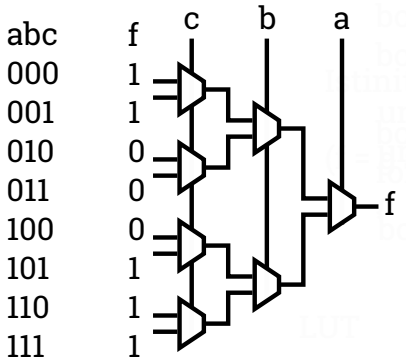
- Prostor $\Theta(1)$ umesto $\Theta(n)$
- Vreme $\Theta(1)$ umesto $\mathcal{O}(n)$

(za 6 promenljivih, $64\times$ manji prostor i do $64\times$ kraće vreme; u praksi još veća razlika u vremenu, zbog smanjenog broja pristupa memoriji)

Jedna fizička realizacija Bulovih funkcija

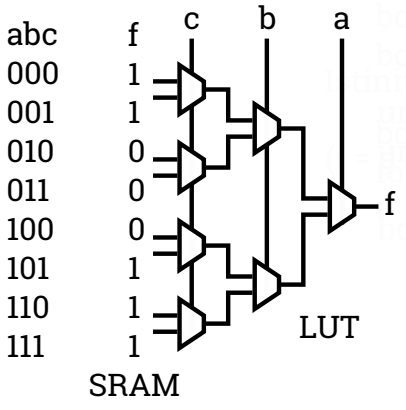
f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

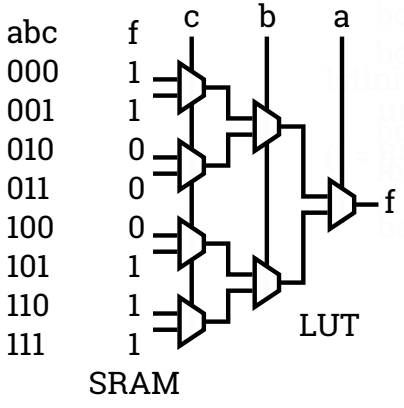


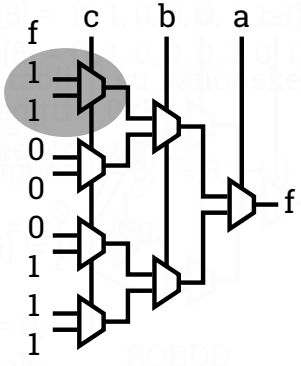
Lukap tabela (eng. Look-Up Table (LUT)): osnova FPGA kola

f = a'b'c' + a'b'c + ab'c + abc' + abc



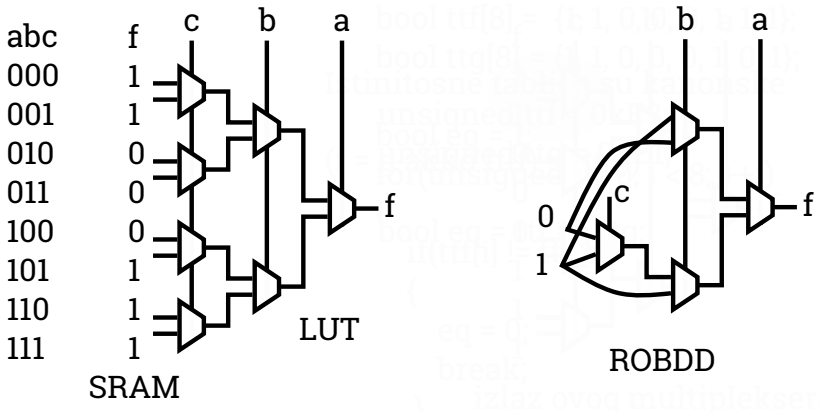
f = a'b'c' + a'b'c + ab'c + abc' + abc





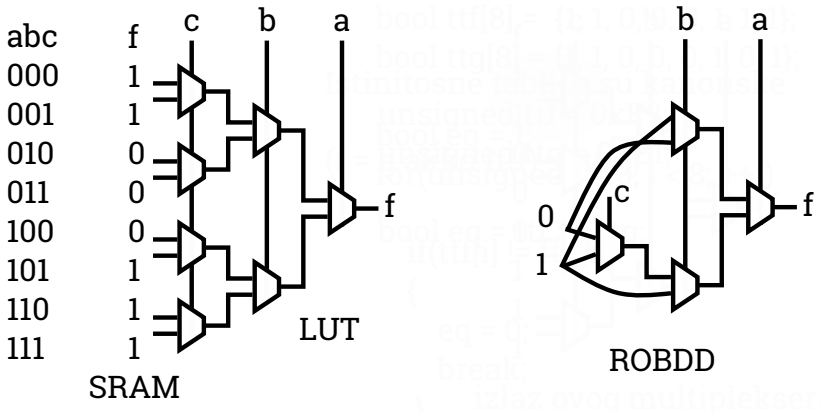
izlaz ovog multipleksera je uvek 1 (ne zavisi od c)

f = a'b'c' + a'b'c + ab'c + abc' + abc



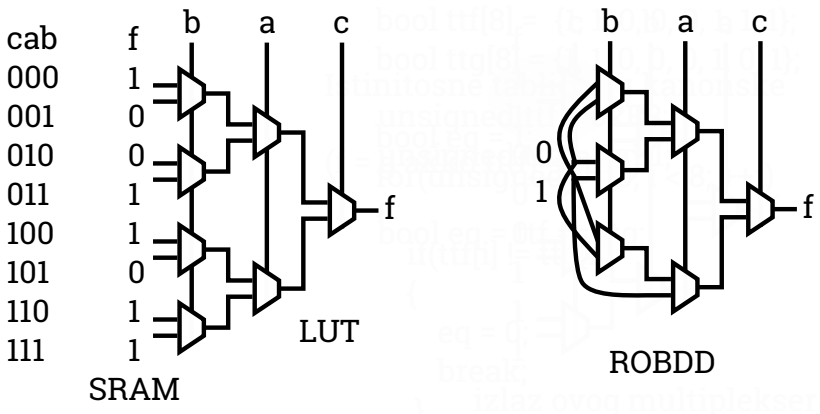
ROBDD-ovi su kanonski, ali samo za dati poredak promenljivih

f = a'b'c' + a'b'c + ab'c + abc' + abc



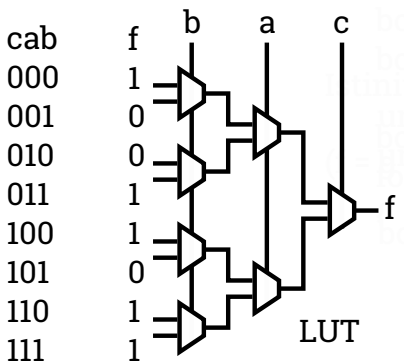
ROBDD-ovi su kanonski, ali samo za dati poredak promenljivih

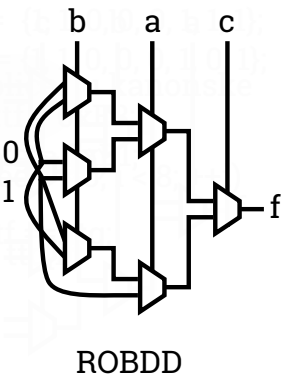
f = a'b'c' + a'b'c + ab'c + abc' + abc



Ako postoji putanja između 1 i f, formula je zadovoljiva

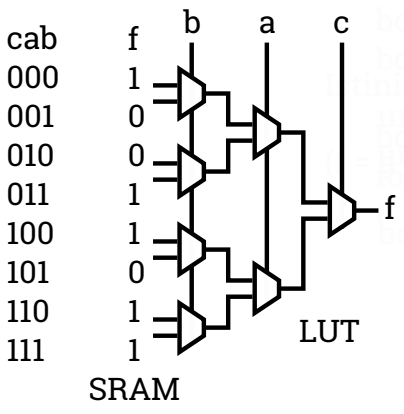
f = a'b'c' + a'b'c + ab'c + abc' + abc

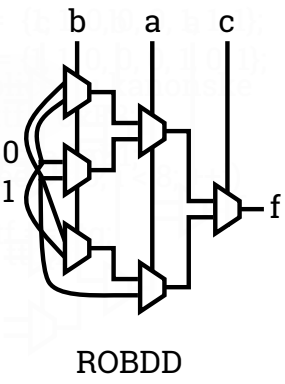




ROBDD može biti eksponencijalno velik, ali u praksi često nije

f = a'b'c' + a'b'c + ab'c + abc' + abc





Šenonov razvoj (eng. Shannon's Expansion)

$f = x'f(x = 0) + xf(x = 1)$

$f(x = 0)$ i $f(x = 1)$ su negativni i pozitivni kofaktor funkcije f spram promenljive x

Rekurzivna primena Šenonovog razvoja $\implies$ binarno stablo odluke

Parcijalni izvodi Bulovih funkcija

$\frac{\partial f}{\partial x} = f(x = 0) \oplus f(x = 1)$

Ukoliko je $\frac{\partial f}{\partial x} = 1$, f zavisi od promenljive x (promena vrednosti x dovodi do promene vrednosti f) i kažemo da x ulazi u **OSNOVU** funkcije f (eng. **SUPPORT**)

Skupovi uključenja i isključenja

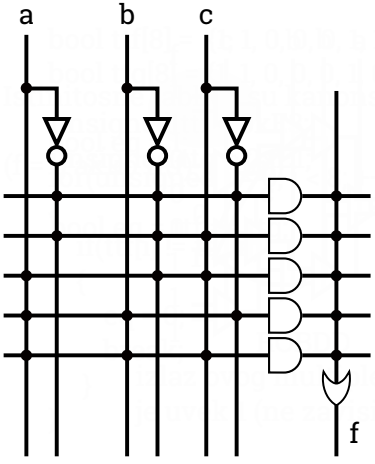
Sve dodele vrednosti ulaznim promenljivama za koje je funkcija 1 spadaju u skup uključenja (eng. **ON-SET**)

Sve dodele vrednosti ulaznim promenljivama za koje je funkcija 0 spadaju u skup isključenja (eng. **OFF-SET**)

Dvostepena logika (eng. TWO-LEVEL LOGIC): direktna implementacija ON-set-a

f = a'b'c' + a'b'c + ab'c + abc' + abc

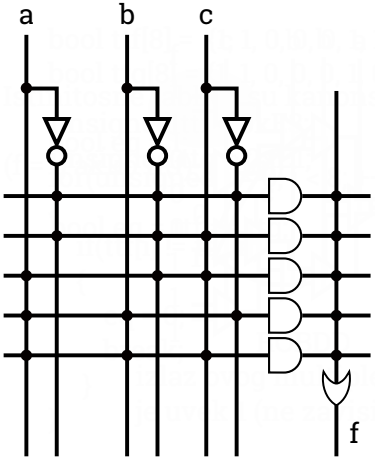
abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1



Cena je proporcionalna zbiru broja ulaza svih kapija. Kako da je minimizujemo?

f = a'b'c' + a'b'c + ab'c + abc' + abc

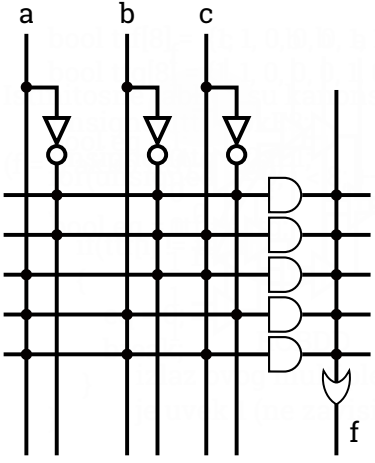
abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1



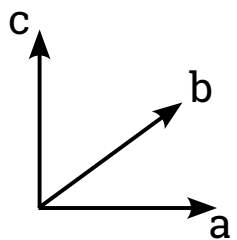
Intuicija: tražimo proizvode koji se razlikuju samo u polarnosti jednog literala

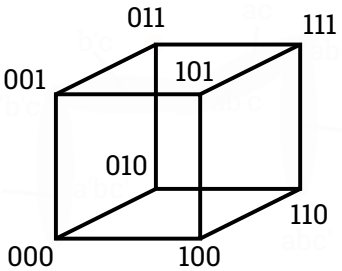
f = a'b'c' + a'b'c + ab'c + abc' + abc

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1

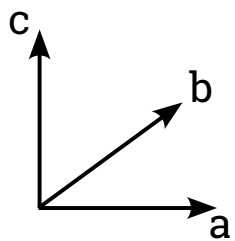


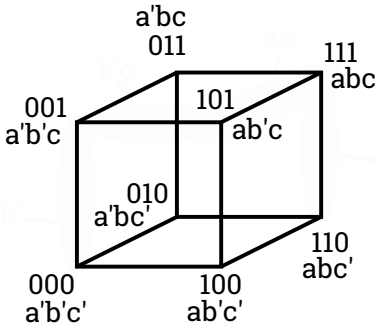
ON-set u prostoru: funkciju je moguće predstaviti kao skup temena hiperkocke





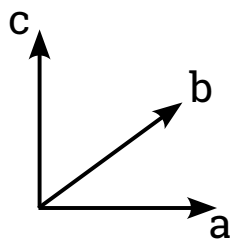
ON-set u prostoru: funkciju je moguće predstaviti kao skup temena hiperkocke

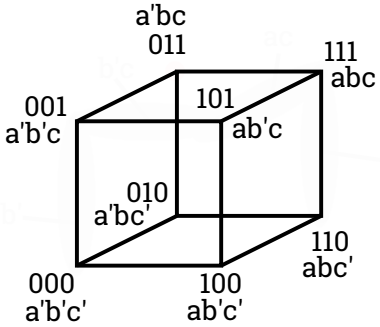




ON-set u prostoru: funkciju je moguće predstaviti kao skup temena hiperkocke

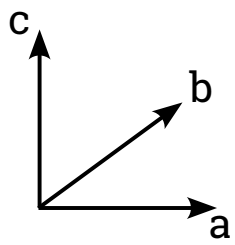
f = a'b'c' + a'b'c + ab'c + abc' + abc

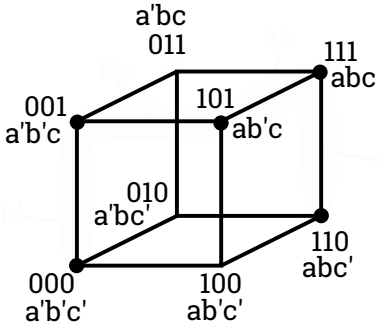




ON-set u prostoru: funkciju je moguće predstaviti kao skup temena hiperkocke

f = a'b'c' + a'b'c + ab'c + abc' + abc





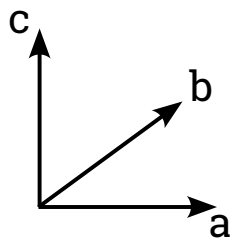
Implikante

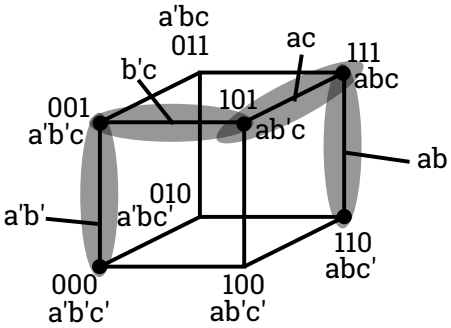
Implikanta je hiperkocka manje dimenzije koja ne sadrži temena iz OFF-set-a (u 3D može biti teme, ivica, ili stranica kocke)

Implikanta je prosta (eng. **Prime Implicant**) ako se ne sadrži ni u jednoj drugoj implikanti

Proste implikante funkcije f

f = a'b'c' + a'b'c + ab'c + abc' + abc





Pokrivanje (eng. Cover)

Pokrivanje je skup implikanti takav da se svako teme iz ON-set-a sadrži barem u jednoj implikanti u tom skupu

Minimalno pokrivanje (eng. Minimum Cover)

Minimalno pokrivanje je pokrivanje sa najmanjim mogućim ukupnim brojem ulaza u kapije kola u odgovarajućoj dvostepenoj realizaciji

Svaka implikanta u pokrivanju = jedna I-kapija

Što je implikanta veća, broj ulaza I-kapije je manji

Što je manje implikanata u pokrivanju, broj ulaza u ILI-kapiju u drugom stepenu je manji

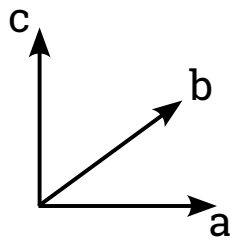
Kvajnova teorema (eng. Quine's theorem)

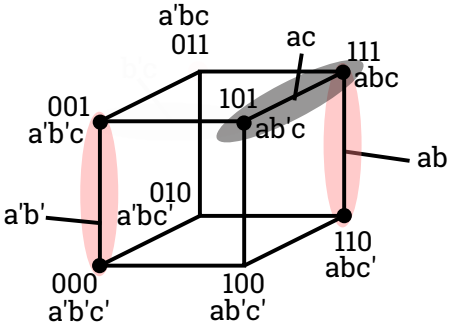
Postoji minimalno pokrivanje u kom je svaka implikanta prosta

Posledica: dovoljno je da posmatramo samo proste implikante

Jedno minimalno pokrivanje funkcije f

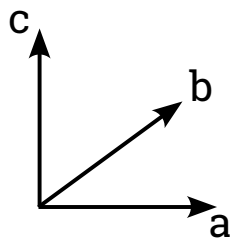
f = a'b'c' + a'b'c + ab'c + abc' + abc

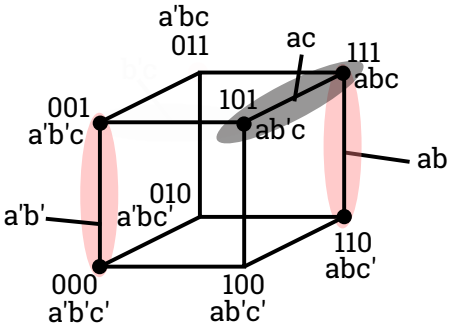




Primetimo da je teme 000 moguće pokriti jedino prostom implikantom $a'b'$

f = a'b'c' + a'b'c + ab'c + abc' + abc





Neophodne implikante

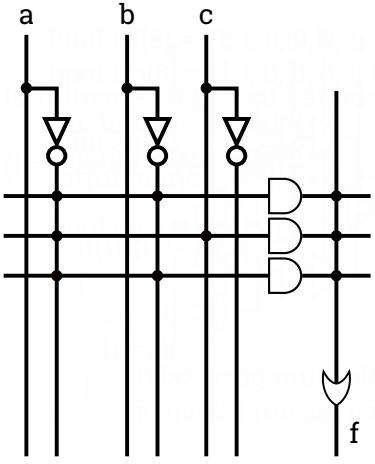
Prema tome, ona mora učestvovati u svakom minimalnom pokrivanju prostim implikantama, te je nazivamo neophodnom (eng. **ESSENTIAL**)

Implikanta *ab* je takođe neophodna

Ekvivalentna dvostepena realizacija (6 umesto 15 ulaza)

f = a'b'c' + a'b'c + ab'c + abc' + abc = ab + ac + a'b'

abc	f
000	1
001	1
010	0
011	0
100	0
101	1
110	1
111	1



Nepotpuno specificirane funkcije

DC-set

Za neke kombinacije promenljivih nije nam bitno koja će biti vrednost funkcije

Na primer, za 1-hot 2:1 multiplekser, znamo da se kombinacije selekcionih ulaza 00 i 11 nikada neće pojaviti, pa ne moramo voditi računa o tome šta će funkcija za njih izbaciti

Da bismo to opisali, proširujemo $\mathbb{B}$ trećim simbolom $*$, koji označava bilo nulu bilo jedinicu

DC-set

U primeru 1-hot 2:1 multipleksera imamo sledeću istinitosnu tablicu

a	s_a	b	s_b	f
0	0	0	0	*
0	0	0	1	0
0	0	1	0	*
0	0	1	1	1
0	1	0	0	0
0	1	0	1	*
0	1	1	0	0
0	1	1	1	*
1	0	0	0	*
1	0	0	1	0
1	0	1	0	*
1	0	1	1	1
1	1	0	0	1
1	1	0	1	*
1	1	1	0	1
1	1	1	1	*

DC-set

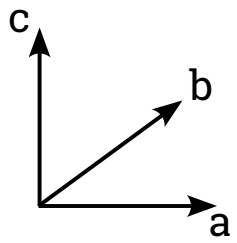
Kombinacije vrednosti promenljivih koje funkcija preslikava na $*$ pripadaju nebitnom skupu (eng. **Don't care set**, DC-set)

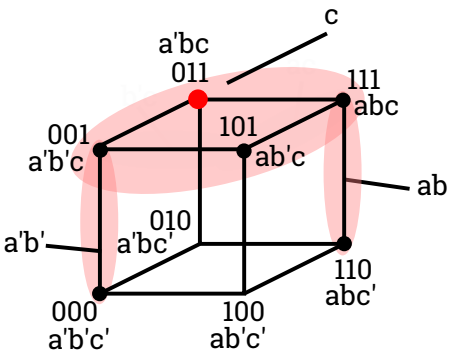
Treću vrednost možemo iskoristiti i za kompresiju istinitosne tablice

a	s_a	b	s_b	f
0	*	*	0	*
*	0	0	1	0
*	0	1	1	1
0	1	*	0	0
1	1	*	0	1

Primer minimizacije prethodne funkcije sa DC-em

f = a'b'c' + a'b'c + ab'c + abc' + abc





Algoritam Kvajna i Meklaskog

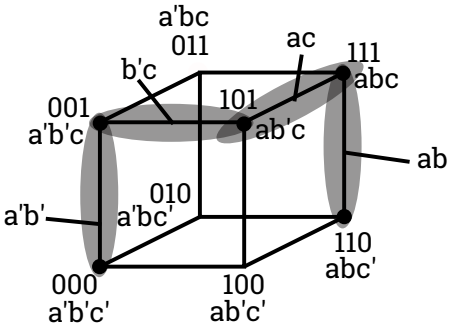
Jednostavnija definicija minimalnosti

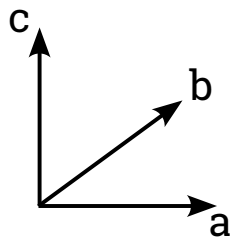
Često korišćena, jednostavnija definicija minimalnosti pokrivanja je: pokrivanje sa najmanjim brojem implikanti

U nastavku ćemo koristiti tu definiciju

Kako da odaberemo proste implikante?

f = a'b'c' + a'b'c + ab'c + abc' + abc



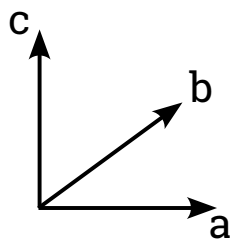


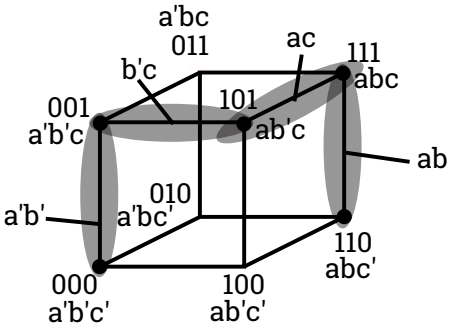
Matrica pokrivanja

Posmatramo koja implikanta pokriva koja temena (eng. **MINTERM**)

ab pokriva abc i abc’

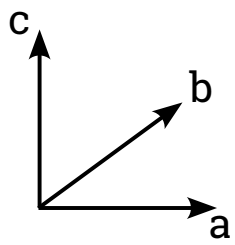
f = a'b'c' + a'b'c + ab'c + abc' + abc

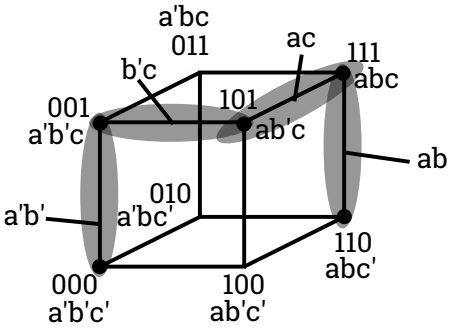




ac pokriva abc i ab’c

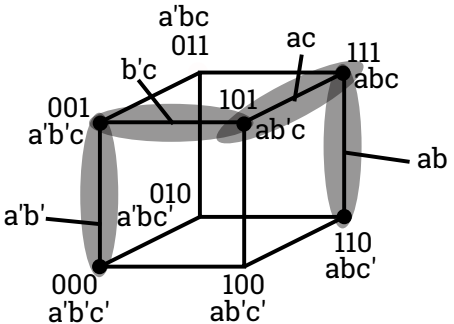
f = a'b'c' + a'b'c + ab'c + abc' + abc

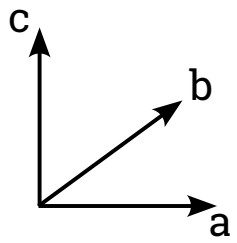




a’b’ pokriva a’b’c’ i a’b’c

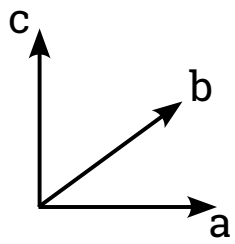
f = a'b'c' + a'b'c + ab'c + abc' + abc

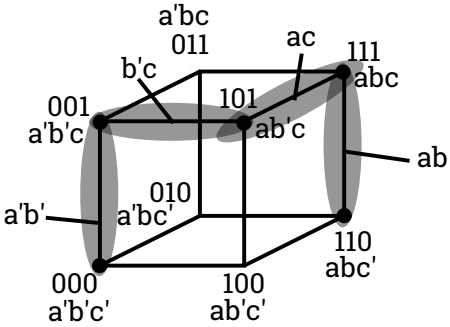




b’c pokriva a’b’c i ab’c

f = a'b'c' + a'b'c + ab'c + abc' + abc





Matrica pokrivanja

To možemo zapisati kao matricu: kolone su implikante, vrste su temena, 1 označava pokrivanje:

	ab	ac	a'b'	b'c
a'b'c'	0	0	1	0
a'b'c	0	0	1	1
ab'c	0	1	0	1
abc'	1	0	0	0
abc	1	1	0	0

Dominacija vrsta

Ako je skup implikanata koje pokrivaju vrstu $i \subseteq$ skupa implikanata koje pokrivaju vrstu i', kažemo da vrsta i' dominira nad vrstom i, odnosno, da je vrsta i dominirana od strane vrste i'

Pokrivanje dominirane vrste implicira pokrivanje dominantne vrste, tako da dominantnu možemo zanemariti

Dominacija vrsta

	ab	ac	a'b'	b'c
a'b'c'	0	0	1	0
~~a'b'c~~	~~0~~	~~0~~	~~1~~	~~1~~
ab'c	0	1	0	1
abc'	1	0	0	0
~~abc~~	~~1~~	~~1~~	~~0~~	~~0~~

Minimizovana matrica

	ab	ac	a'b'	b'c
a'b'c'	0	0	1	0
ab'c	0	1	0	1
abc'	1	0	0	0

Dominacija kolona

Ako je skup temena koja pokriva kolona $j \subseteq$ skupa temena koje pokriva kolona j', kažemo da kolona j' dominira nad kolonom j, odnosno, da je kolona j dominirana od strane kolone j'

Izbor dominantne kolone implicira pokrivanje svih temena koja bi pokrila dominirana kolona, pa dominiranu možemo zanemariti

Minimizovana matrica

	ab	ac	a'b'
a'b'c'	0	0	1
ab'c	0	1	0
abc'	1	0	0

Neophodne implikante

U matrici pokrivanja, kolone neophodnih implikanti se seku sa bar jednom vrstom koja osim u datoj implikanti nema ni jednu drugu jedinicu

Sve neophodne implikante moramo uvrstiti u pokrivanje, a zatim možemo ukloniti iz matrice pokrivanja njihove kolone i sve vrste u kojima imaju jedinicu

Nakon poslednje minimizacije, sve implikante su neophodne

	ab	ac	a'b'
a'b'c'	0	0	1
ab'c	0	1	0
abc'	1	0	0

Prema tome, jedno minimalno pokrivanje je ab + ac + a'b'

Minimizacija matrice pokrivanja

1. odbaci sve dominantne vrste
2. odbaci sve dominirane kolone
3. prihvati sve neophodne kolone i uprosti matricu
4. ponavljaj dok ima napretka

Šta da radimo ako iteratvna minimizacija ne ukloni sve vrste?

Primer

α	β	γ	δ	ϵ	ζ	η	θ
0	0	0	0	0	0	1	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1
1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0

Iterativno uvećanje

Dodajemo redom implikantu po implikantu, dok ne pokrijemo sva temena

Nakon svakog dodavanja, vršimo minimizaciju matrice

Iterativno uvećanje

α	β	γ	δ	ϵ	ζ	η	θ
0	0	0	0	0	0	1	1
~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~
0	1	0	0	0	0	0	1
~~1~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
0	1	0	1	0	0	0	0
~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~
0	1	0	0	0	1	0	0
0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0

α

Uklanjamo dominirane kolone γ i η

Iterativno uvećanje

β	δ	ϵ	ζ	θ
0	0	0	0	1
1	0	0	0	1
1	1	0	0	0
1	0	0	1	0
0	1	0	0	0
0	0	1	1	0

α

Uklanjamo dominantne vrste, 2 i 3

Iterativno uvećanje

β	δ	ϵ	ζ	θ
~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
1	0	0	1	0
~~0~~	~~1~~	~~0~~	~~0~~	~~0~~
0	0	1	1	0

$\alpha + \delta + \theta$

Dodajemo neophodne kolone δ i θ

Iterativno uvećanje

β	ϵ	ζ
1	0	1
0	1	1

$\alpha + \delta + \theta$

Odbacujemo dominirane kolone, β i ϵ

Iterativno uvećanje

$$\begin{array}{c} \zeta \\ \hline 1 \\ 1 \end{array}$$

Kolona ζ je neophodna, pa dobijamo pokrivanje:

$\alpha + \delta + \theta + \zeta$

Da li je to pokrivanje minimalno?

Dve vrste minimalnosti

Pokrivanje može biti globalno minimalno (ili samo minimalno, eng. **MINIMUM**) ako ne postoji pokrivanje sa manjim brojem implikanti i lokalno minimalno (eng. **MINIMAL**) odnosno neredundantno (eng. **IRREDUNDANT**) ako odbacivanje bilo koje implikante iz datog pokrivanja otkriva neko teme iz ON-set-a

Da li je $\alpha + \delta + \theta + \zeta$ neredundantno pokrivanje?

α	δ	ζ	θ
0	0	0	1
1	0	0	0
0	0	0	1
1	0	0	0
0	1	0	0
1	0	0	0
0	0	1	0
0	1	0	0
0	0	1	0

Sve implikante su neophodne $\implies$ jeste

Da li je $\alpha + \delta + \theta + \zeta$ neredundantno pokrivanje?

α	δ	ζ	θ
0	0	0	1
1	0	0	0
0	0	0	1
1	0	0	0
0	1	0	0
1	0	0	0
0	0	1	0
0	1	0	0
0	0	1	0

No, da li je i globalno minimalno (da li smo umesto α mogli da dodamo neku drugu implikantu i dobijemo pokrivanje sa tri implikante)?

Donje granice

α	β	γ	δ	ϵ	ζ	η	θ
0	0	0	0	0	0	1	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1
1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0

Kardinalnost pokrivanja je uvek $\geqslant$ od kardinalnosti najvećeg skupa nezavisnih vrsta $\implies$ našli smo minimalno pokrivanje

Tačni algoritmi

U opštem slučaju, ovakav pristup, bez uklanjanja ranije dodatih kolona ne bi doveo do optimuma

Da bismo to garantovali, moramo izvršiti potpunu pretragu prostora (eng. EXHAUSTIVE SEARCH)

Brute force + binary search

1. Redom izlistavamo kombinacije k implikanti bez ponavljanja
2. Za svaku proveravamo da li je pokrivanje
3. Ukoliko je pokrivanje nađeno, ograničavamo interval binarne pretrage k na levu polovinu
4. Ukoliko nije, potrebno nam je više implikanti, te ograničavamo na desnu polovinu

Moguća je i linearna pretraga umesto binarne

Prednosti ovog algoritma?

Prednosti ovog algoritma?

- Lak za implementaciju
- Lak za paralelizaciju (možemo testirati više kombinacija istovremeno)

Mane ovog algoritma?

Mane ovog algoritma?

- Ne koristimo tehnike minimizacije matrice pokrivanja
- Ako smo prinuđeni da razmatramo veće kombinacije, ne koristimo deo posla koji je već obavljen prilikom razmatranja manjih

U prethodnom primeru, čak i da smo unapred znali da je minimalna kardinalnost pokrivanja 4, morali bismo da pretražimo i do $\binom{8}{4} = 70$ kombinacija dok bismo našli pokrivanje

Kako da rešimo ove probleme?

Enumeracija kombinacija

Neka imamo skup $\{A, B, C, D, E\}$

Na prvoj poziciji mogu biti svi elementi: $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}$

Pošto ne želimo ponavljanja, na drugoj poziciji mogu biti svi elementi osim prvog: $\{A, B\}, \{A, C\}, \{A, D\}, \ldots, \{B, A\}, \{B, C\}\ldots$

Šta je ovde problem?

Enumeracija kombinacija

Da bismo izbegli enumeraciju permutacija istih kombinacija, uređujemo polazni skup elemenata (A, B, C, D, E) i uvodimo uslov da svaka kombinacija može biti proširena samo elementima koji su veći od poslednjeg dodatog

Ukupan algoritam za enumeraciju

1. neka stek S sadrži parcijalne kombinacije (u početku samo praznu kombinaciju)
2. Dokle god stek S nije prazan, popuj poslednju dodatu parcijalnu kombinaciju, c i ispiši je
3. Za svaki element e polaznog skupa $>$ od poslednjeg elementa iz c, dodaj $c \cup \{e\}$ na stek

Gde možemo da dodamo testiranje pokrivanja?

1. neka stek S sadrži parcijalne kombinacije (u početku samo praznu kombinaciju)
2. Dokle god stek S nije prazan, popuj poslednju dodatu parcijalnu kombinaciju, c i ispiši je
3. Za svaki element e polaznog skupa $>$ od poslednjeg elementa iz c, dodaj $c \cup \{e\}$ na stek

Gde možemo da dodamo testiranje pokrivanja?

1. neka stek S sadrži parcijalne kombinacije (u početku samo praznu kombinaciju)
2. Dokle god stek S nije prazan, popuj poslednju dodatu parcijalnu kombinaciju, c i ~~ispiši~~ testiraj je
3. Za svaki element e polaznog skupa $>$ od poslednjeg elementa iz c, dodaj $c \cup \{e\}$ na stek

Gde možemo da dodamo minimizaciju matrice pokrivanja?

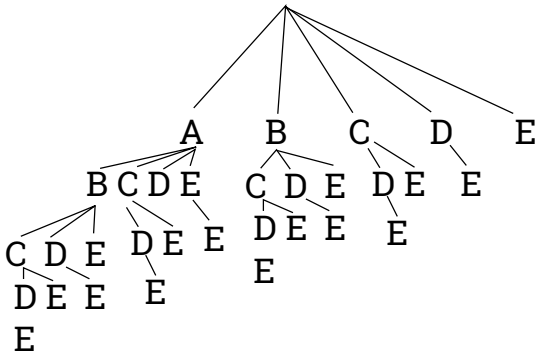
1. neka stek S sadrži parcijalne kombinacije (u početku samo praznu kombinaciju)
2. Dokle god stek S nije prazan, popuj poslednju dodatu parcijalnu kombinaciju, c i testiraj je
3. Za svaki element e polaznog skupa $>$ od poslednjeg elementa iz c, dodaj $c \cup \{e\}$ na stek

Gde možemo da dodamo minimizaciju matrice pokrivanja?

1. neka stek S sadrži parcijalne kombinacije (u početku samo praznu kombinaciju)
2. Dokle god stek S nije prazan, popuj poslednju dodatu parcijalnu kombinaciju, c
3. Minimizuj matricu pokrivanja. Ako je ostala prazna, pokrivanje je pronađeno, zabeleži ga
4. Za svaku preostalu kolonu matrice pokrivanja $e >$ od poslednjeg elementa iz c, dodaj $c \cup \{e\}$ na stek

Najmanje zabeleženo pokrivanje je rešenje

Ovakvu vrstu pretrage je moguće predstaviti kao stablo



Pošto pri svakom koraku donosimo odluku o tome koju kombinaciju da proširimo i kojim elementom (razne heuristike su moguće; na primer, implikantom koja pokriva najveći broj temena), nazivamo je grananjem (eng. **BRANCHING**)

Bounding

U opštem slučaju moramo pretražiti svih 2^n kombinacija. Međutim, dok budemo vršili pretragu, na primer u DFS poretku, u nekom trenutku ćemo naići na pokrivanje kardinalnosti k. Sva pokrivanja sa kardinalnošću $\geqslant k$ možemo zanemariti. To nazivamo orezivanjem stabla pretrage (eng. PRUNING).

Bounding

Osnovno orezivanje se naprosto svodi na prestanak proširenja parcijalnih rešenja čija bi kardinalnost tim proširenjem dosegla kardinalnost već poznatog tekućeg (eng. **INCUMBENT**) rešenja

No, ako imamo mogućnost da predvidimo da će parcijalno rešenje sa cenom i zahtevati uvećanje cene od još najmanje j, i ako je $i + j \geqslant k$, možemo unapred izvršiti orezivanje

Ukoliko je procena j uvek donje ograničenje, nećemo promašiti stvarni optimum, a što je bliže stvarnoj dodatnoj ceni, orezivanje će biti efikasnije

Bounding

Ova tehnika se naziva ograničavanje (eng. **BOUNDING**) a ukupan način pretrage **BRANCH & BOUND**

Napomena: branch and bound nije isto što i branch and cut (pruning i cutting nisu isto)

Unate cover

Problem pokrivanja sa kojim smo se susreli nazivamo **UNATE COVER**

Kao što smo ranije napomenuli, efikasna donja granica cene je kardinalnost najvećeg skupa nezavisnih vrsta

Međutim, njeno računanje zahteva bojenje grafa preseka vrsta i jednako je teško kao i računanje pokrivanja

Ipak, moguće je izvršiti aproksimaciju (na primer, nasumično dodajemo vrste nezavisne od već dodatih, dok ih ima; više startova može dovesti do boljih rezultata)

Zaključak (neki opšti recepti za rešavanje teških problema)

- Rešenje gradimo inkrementalno (Python itertools vraća sve kombinacije, ali nam to ne bi pomoglo)
- Čim naučimo da dalje proširenje nema smisla, prestajemo
- Prvo uzimamo sve što nam je neophodno i redukujemo prostor pretrage na osnovu toga
- U svakom koraku se oslobađamo svega što ne moramo eksplicitno da razmatramo

Analogija: ako je polica preteška, pre nego što odustanemo, pokušajmo da uklonimo knjige

Pohlepan pristup

Nekad nemamo vremena da tražimo optimum (uključujući i ako nam je potreban dobar početni prag za orezivanje u Branch and Bound-u)

Tada rešenje gradimo inkrementalno, ali na pohlepan način, bez vraćanja na razmatranje ranijih odluka

Varijanta 1

1. Uzmi implikantu koja pokriva najviše temena
2. Uprosti matricu pokrivanja
3. Ako je prazna, završi, ako ne, vrati se na 1)

Kako možemo da poboljšamo ovo rešenje?

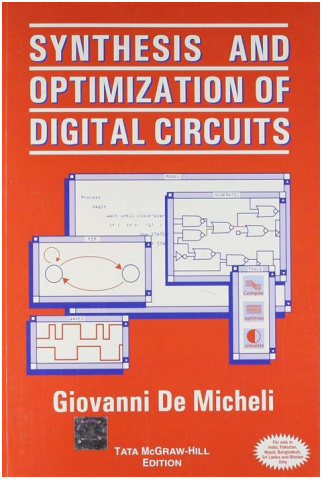
1. Uzmi implikantu koja pokriva najviše temena
2. Uprosti matricu pokrivanja
3. Ako je prazna, završi, ako ne, vrati se na 1)

Varijanta 2

1. Počni od praznog parcijalnog rešenja
2. Za sve preostale implikante, kopiraj parcijalno rešenje i proširi ga datom implikantom
3. Zameni parcijalno rešenje proširenjem koje je ostavilo najmanje nepokrivenih temena (i najmanje implikanti) nakon uprošćenja matrice pokrivanja

Sada smo od linearnog broja generisanih proširenja u najgorem slučaju došli do kvadratnog, ali je potencijalno kvalitet dobijenog rešenja veći

Današnje predavanje je u velikoj meri bilo zasnovano na knjizi prof. De Mikelija



Dvostepena logika je nekada bila dominantan način realizacije kola

https://en.wikipedia.org/wiki/Programmable_logic_array

Programmable logic array

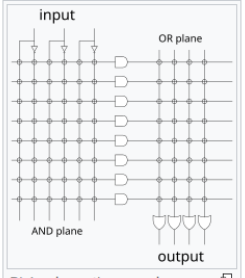
9 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Programmable Array Logic.

A **programmable logic array** (**PLA**) is a kind of programmable logic device used to implement combinational logic circuits. The PLA has a set of programmable AND gate planes, which link to a set of programmable OR gate planes, which can then be conditionally complemented to produce an output. It has 2^N AND gates for N input variables, and for M outputs from the PLA, there should be M OR gates, each with programmable inputs from all of the AND gates. This layout allows for many logic functions to be synthesized in the sum of products canonical forms.



PLA schematic example

PLAs differ from programmable array logic devices (PALs and GALs) in that both the AND and OR gate planes are programmable. PAL has programmable AND gates but fixed OR gates [*clarification needed*]

No, zbog fizičke implementacije to odavno nije slučaj

Višestepenu logiku ćemo raditi na sledećem času