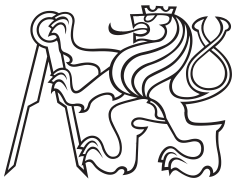
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra telekomunikační techniky

Online nástroj pro minimalizaci logických funkcí metodou Karnaughových map a Quine-McCluskey

Vít Kodat

Školitel: Ing. Pavel Lafata, Ph.D.
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kodat** Jméno: **Vít** Osobní číslo: **483891**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Online nástroj pro minimalizaci logických funkcí metodou Karnaughových map a Quine-McCluskey

Název bakalářské práce anglicky:

Online tool for logic function minimization based on Karnaugh map and Quine-McCluskey algorithm

Pokyny pro vypracování:

Seznamte se s metodou minimalizace logických funkcí pomocí Karnaughových map a algoritmu Quine-McCluskey a s realizací logických funkcí pomocí základních log. hradel. Navrhněte a implementujte online nástroj pomocí jazyka JavaScript (případně Java), který bude sestávat z přehledného uživatelského rozhraní pro zadání logické funkce, z vlastní implementace minimalizace pomocí výše uvedených metod a grafického i algebraického výstupu řešení. Další upřesnění funkcionalit nástroje bude upřesněno na konzultaci s vedoucím zadání. Online nástroj bude posléze umístěn na webový server.

Seznam doporučené literatury:

[1] Lafata, P. - Hampl, P. - Pravda, M.: Digitální technika. 1. vyd. Praha: Česká technika - nakladatelství ČVUT, 2011. 164 s.
[2] T. K. Jain, D. S. Kushwaha and A. K. Misra, "Optimization of the Quine-McCluskey Method for the Minimization of the Boolean Expressions," Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08), Gosier, 2008, pp. 165-168, doi: 10.1109/ICAS.2008.11.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Pavel Lafata, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji vedoucímu Ing. Pavlu Lafatovi Ph.D za cenné rady, nápady a vedení při realizaci mé bakalářské práce.

Prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci vypracoval samostatně pod vedením vedoucího práce a že jsem uvedl veškeré použité zdroje v souladu s Metodickým pokynem o dodržování etických principů při tvorbě vysokoškolských prací.

V Praze, 21. května 2021

Abstrakt

Bakalářská práce se zabývá tématem minimalizace logických funkcí a vytvořením online nástroje pro minimalizaci těchto funkcí. Nástroj je vytvořen ve značkovacím jazyce HTML a programovacím jazyce JavaScript a je umístěn na webový server. Minimalizace je řešena implementací Quine-McCluskey algoritmu.

Klíčová slova: Quine-McCluskey, algoritmus, logická funkce, minimalizace, booleova algebra

Školitel: Ing. Pavel Lafata, Ph.D.

Abstract

Bacheor thesis deals with minimalization of logic functions and develops online tool for minimalization of those functions. The tool is progarammed in markup language HTML and programming language JavaScript. Tool is uploaded to web server. Minimalization of functions is solved by using Quine-McCluskey algorithm.

Keywords: Quine-McCluskey, algorithm, logical function,minimization,boolean algebra

Title translation: Online tool for logic function minimization based on Karnaugh map and Quine-McCluskey algorithm

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Digitální elektronické systémy jsou každodenní součástí životů všech lidí na planetě. Ať už se bavíme o digitálních hodinkách s relativně jednoduchými integrovanými obvody, nebo o mobilních telefonech s nejmodernějšími mikroprocesory, každé digitální zařízení ve svém nitru provádí logické funkce. Výhodnou implementací logických funkcí můžeme díky použití menšího množství logických hradel ušetřit finanční náklady na výrobu zařízení i místo na deskách plošných spojů.

Cílem práce je vytvořit online nástroj, který zjistí minimální tvar zadané logické funkce pomocí Karnaughových map a Quine-McCluskey (dále jen QmC) algoritmu a tento tvar vypíše v minimální disjunktní formě i minimální konjunktní formě. Online nástroj se bude skládat z přehledného uživatelského rozhraní pro zadání logické funkce a pro zobrazení výsledku minimalizace, a z vlastní implementace minimalizace pomocí výše uvedených metod. Online nástroj bude umístěn na webovém serveru.

Vzhledem k bodu zadání, že se má jednat o online nástroj, bude nejvýhodnější tento nástroj naprogramovat jako webovou aplikaci spustitelnou na kterémkoli webovém prohlížeči. Webové prohlížeče dnes existují prakticky na všech operačních systémech, takže bude možné nástroj spustit na chytrých mobilních telefonech i osobních počítačích.

Webová aplikace bude realizována kombinací značkovacího jazyka HTML a JavaScriptu. Výhoda tohoto řešení spočívá v distribuci výpočetní zátěže při minimalizaci logické funkce na připojeného uživatele, a tudíž není minimalizací logické funkce vytěžován webový server. Navíc vzhledem k podstatě této práce je zdrojový kód zcela veřejný a obavy o zkopírování zdrojového kódu jsou irelevantní.

V příští kapitole "Teoretický rozbor", bude popsán teoretický základ nezbytně potřebný pro implementaci řešení online nástroje.

Další kapitola "Řešení" popisuje způsob minimalizace logické funkce v programovacím jazyce JavaScript.

Kapitola 2
Teoretický rozbor

Následující podkapitoly popisují nezbytný teoretický základ, který dále využijeme při implementaci online nástroje. Základní stavební kámen teorie minimalizace logických funkcí tvoří binární číselná soustava.

2.1 Binární číselná soustava

Binární číselná soustava reprezentuje čísla pomocí 2 číslic - 0 a 1. Jedná se o poziční číselnou soustavu, tudíž pozice číslice v řetězci číslic udává její hodnotu. Kapacita binární číselné soustavy je rovna 2^N, kde N je rovno počtu číslic. Znamená to, že například pomocí 4 binárních číslic je možno v binární číselné soustavě vyjádřit 2^4 čísel. [1] [2]

Výhodou binární číselné soustavy je její jednoduchá technická realizace výpočetními systémy. Vzhledem k tomu, že jednotlivé číslice mohou nabývat pouze 2 stavů, lze tyto stavy reprezentovat jakýmkoliv 2 stavovým signálovým prvkem. Základem elektronických systémů pracujících s binárními daty jsou pak tranzistory, jejichž výstupní brána z hlediska průchodnosti elektrickým proudem nabývá 2 stavů: průchozí a neprůchozí pro elektrický proud.

Důležitým pojmem je logická proměnná. Ta může v binární logice nabývat 2 stavů 0 a 1.

2.2 Logická funkce

Logická funkce jednoznačně popisuje vztah mezi vstupní množinou hodnot a výstupní množinou hodnot. Vstupní množinu hodnot představuje vektor vstupních proměnných. Každá vstupní proměnná může nabývat 2 logických stavů. Vektor vstupních proměnných nabývá celkem 2^n stavů, kde n je počet prvků vektoru.Výstupní množinu hodnot tvoří pouze 2 logické hodnoty, 0 a 1.

V praxi jsou logické funkce hojně využívány pro popis chování digitálních obvodů. Digitální obvod, který má 2 vstupní vodiče a 1 výstupní vodič, lze plně popsat logickou funkcí, přičemž její jednotlivé vstupní proměnné nabývají stejné hodnoty, jako příslušné vstupní vodiče.

Existuje více způsobů, jak logickou funkci popsat. Vzhledem k tomu, že definiční obor funkce je konečná množina, je možné funkci popsat pravdi-

vostní tabulkou, ve které pro každý možný stav vstupního vektoru zapíšeme příslušnou výstupní hodnotu funkce. Pravdivostní tabulka 2.1 popisuje logickou funkci, jejíž vstupní vektor $\boldsymbol{v}$ má délku 3 a skládá se ze 3 vstupních proměnných $\boldsymbol{v} = [a\ b\ c]$.

Tabulka 2.1: Příklad pravdivostní tabulky.

N	a	b	c	f
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

Další možností, jak popsat logickou funkci, je vypsáním indexů, ve kterých logická funkce nabývá logické 1. Pokud bychom takto popsali logickou funkci prezentovanou výše, výsledek by byl následující:

$$f = 0, 1, 4, 5.$$

Speciálním příkladem logické funkce je neurčitá logická funkce. Tato funkce pro jeden nebo více stavů vstupního vektoru nemá definovanou výstupní hodnotu. Výstupní hodnota se v zápisu logické funkce pravdivostní tabulkou nahrazuje písmenem X. Logická funkce popsaná v tabulce 2.2 nemá definovanou výstupní hodnotu pro stavy vstupního vektoru $\boldsymbol{v} = [1\ 0\ 0]$ a $\boldsymbol{v} = [0\ 1\ 1]$. Popisem obvodu neurčitou logickou funkcí by jeho výrobce například zdůrazňoval, že není schopen za daných podmínek garantovat konkrétní stav výstupu obvodu.

Tabulka 2.2: Neurčitá logická funkce.

N	a	b	c	f
0	0	0	0	1
1	0	0	1	X
2	0	1	0	0
3	0	1	1	X
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

2.3 Booleova algebra

Pravidla pro manipulaci s logickými proměnnými zavádí Booleova algebra. Booleova algebra používá 3 základní logické operace. Operace AND (logický součin, značka $\cdot$), operace OR (logický součet, značka $+$), a operace NOT (logická negace, značí se linkou nad negovanou proměnnou) [3]. Logické operace AND a OR jsou binární operace, které požadují 2 operandy, tedy 2 logické proměnné. Výsledkem obou těchto operací je logická hodnota. Logická operace NOT je unární, lze ji aplikovat na 1 vstupní proměnnou. Výsledek je logická hodnota. Axiomy Booleovy algebry jsou vypsány v tabulce 2.3. V tabulce a, b i c představují logické proměnné nabívající buď logické 1, nebo logické 0. Z uvedených axiomů lze odvodit vztahy vypsané v tabulce 2.4. [4]

Tabulka 2.3: Axiomy Booleovy algebry.

asociativita	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
komutativita	$a + b = b + a$	$a \cdot b = b \cdot a$
identita	$a + 0 = a$	$a \cdot 1 = a$
distributivita	$a + (b \cdot c) = (a + b) \cdot (a + c)$	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
vyloučení třetího	$a + \bar{a} = 1$	$a \cdot \bar{a} = 0$

Tabulka 2.4: Odvozené vztahy platné v Booleově algebře.

idempotence prvků	$a \cdot \bar{a} = 0$	$a + \bar{a} = 1$
agresivita 0 a 1	$a + 1 = 1$	$a \cdot 0 = 0$
absorbce	$a + (a \cdot b) = a$	$a \cdot (a + b) = a$
DeMorganovy zákony	$\overline{a + b} = \bar{a} \cdot \bar{b}$	$\overline{a \cdot b} = \bar{a} + \bar{b}$

Pomocí Booleovy algebry je možné popsat logickou funkci, jelikož každá logická funkce lze vyjádřit pomocí logických operací aplikovaných na vstupní proměnné funkce. Každá funkce lze vyjádřit ve tvaru součtu logických součinů (disjunktním tvaru) nebo ve tvaru součinu logických součtů (konjunktním tvaru). [5]

V disjunktním tvaru lze funkci popsat tak, že pro každý stav vstupního vektoru, kdy výstupní hodnota funkce bude logická 1, provedeme logický součin vstupních proměnných. S každou vstupní proměnnou ale musíme provést takovou logickou operaci, aby po jejím aplikování na tuto vstupní proměnnou při daném stavu vstupního vektoru byla hodnota této proměnné logická 1. To znamená, že pokud byla daná vstupní proměnná při daném stavu vstupního vektoru rovna logické 1, nebudeme aplikovat žádnou operaci, a pokud byla proměnná rovna logické 0, aplikujeme operaci negace. Tímto nám vznikne tolik členů, kolik existuje stavů vstupního vektoru, kdy je funkce rovna 1. Tyto členy se nazývají mintermy. Mezi mintermy vložíme logický

součet. Funkci uvedenou v tabulce 2.1 lze tímto způsobem popsat následovně:

$$f(\boldsymbol{v}) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c. \tag{2.1}$$

Konjunktní tvar logické funkce můžeme sestrojit tak, že pro každý stav vstupního vektoru, kdy výstupní hodnota funkce bude logická 0, provedeme logický součet vstupních proměnných. S každou vstupní proměnnou ale musíme provést takovou logickou operaci, aby po jejím aplikování na tuto vstupní proměnnou při daném stavu vstupního vektoru byla hodnota této proměnné logická 0. To znamená, že pokud byla daná vstupní proměnná při daném stavu vstupního vektoru rovna logické 0, nebudeme aplikovat žádnou operaci, a pokud byla proměnná rovna logické 1, aplikujeme operaci negace. Tímto nám vznikne tolik členů, kolik existuje stavů vstupního vektoru, kdy je funkce rovna 0. Tyto členy se nazývají maxtermy. Mezi maxtermy vložíme logický součin. Funkci uvedenou v tabulce 2.1 lze tímto způsobem popsat následovně:

$$f(\boldsymbol{v}) = (a + \bar{b} + c)(a + \bar{b} + \bar{c})(\bar{a} + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c}). \tag{2.2}$$

2.4 Logická hradla, realizace logické funkce digitálními obvody

Využití logické funkce nacházejí hlavně v digitálních elektronických obvodech. Logická hodnota je reprezentována konkrétním parametrem elektrického obvodu. Nejčastěji se k reprezentaci logických hodnot využívá elektrické napětí. Například v logice TTL napěťový interval 0 V až 0,8 V reprezentuje logickou úroveň 0 a napěťový interval 2 V až 5 V reprezentuje logickou úroveň 1.

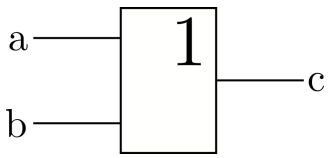
Základní logické operace OR, AND a NOT jsou zásadní pro popis logických funkcí pomocí Booleovy algebry. Při implementaci logické funkce elektronickými obvody tyto operace vykonávají logická hradla. Logická hradla jsou digitální elektronické obvody realizující logické operace. Logická úroveň na výstupu hradla je dána logickými úrovněmi přivedenými na vstup hradla a logickou operací, kterou příslušné hradlo reprezentuje.

Na obrázku 2.1 vidíme schematickou značku hradla OR a dále pravdivostní tabulku tohoto hradla. Nízká logická úroveň na výstupu hradla bude pouze v případě, že oba vstupy hradla budou taktéž vykazovat nízkou logickou úroveň. Při jakémkoliv jiném stavu vstupů hradla bude výstup hradla vykazovat logickou 1. [6]

Logický součin, další základní binární operace, je v elektronických obvodech prováděna hradlem AND. Jeho schématickou značku vidíme na obrázku 2.2. Aby hradlo AND na svůj výstup připojilo vysokou logickou úroveň, musí být všechny jeho vstupy ve stavu logické 1. V opačném případě je na výstupu hradla logická 0. [6]

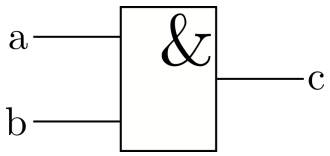
Počet vstupů hradel AND i OR může být neomezeně škálován. Škálovaná hradla se chovají analogicky k jejich dvouvstupovým vzorům. Hradlo OR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1



Obrázek 2.1: Hradlo OR a jeho pravdivostní tabulka

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1



Obrázek 2.2: Hradlo AND a jeho pravdivostní tabulka

vykáže na svém výstupu logickou 1, pokud alespoň 1 z jeho N vstupů bude ve stavu logické 1 a hradlo AND vykáže na svém výstupu logickou 1 v případě, že všechny jeho vstupy budou ve stavu logické 1.

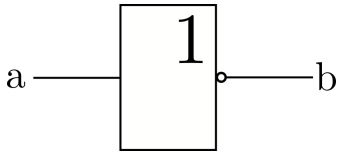
Poslední základní logickou operací je negace. Tu lze obvodově realizovat hradlem NOT, které vidíme na obrázku 2.3. Hradlo má pouze 1 vstup a 1 výstup, na který vykáže logickou 1 v případě, že vstup hradla bude ve stavu logické 0. Po přivedení na vstup hradla vysoké logické úrovně bude výstup hradla ve stavu logické 0. [6]

Kombinací logických hradel je možné realizovat jakoukoliv funkci. Například rovnice 2.1 říká, že provedením logických součinů $\bar{a}\bar{b}\bar{c}$, $\bar{a}\bar{b}c$, $a\bar{b}\bar{c}$, $a\bar{b}c$ a následným logickým součtem dílčích výsledků získáme výstup logické funkce 2.1. Příslušné zapojení hradel je na obrázku 2.4. Jednotlivé logické součiny lze provést pomocí hradel AND se třemi vstupy, kde na každý vstup přivedeme příslušnou logickou proměnnou. Výstupy ze všech hradel AND zapojíme po jednom do vstupů logického hradla OR, čímž bude proveden logický součet logických součinů. Výstup hradla OR se bude chovat podle funkce 2.1. Obvod 2.4 se nazývá dekodér a jeho hlavní využití spočívá v detekování určitých logických stavů sběrnice. Typickým použitím je jeho připojení na adresovou sběrnici, přičemž pokud na sběrnici zaznamená adresu příslušející zařízení připojeném na jeho výstupu, nastaví svůj výstup na logickou 1, čímž aktivuje adresované zařízení. Adresované zařízení může být například RAM.

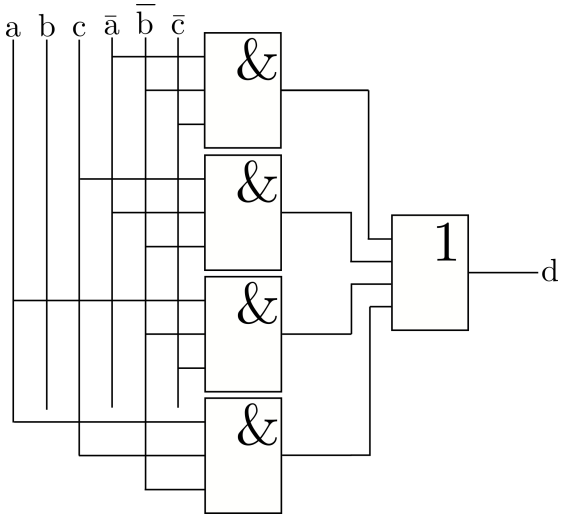
2.5 Minimalizace logických funkcí

V předchozím textu jsme pracovali s logickou funkcí 2.1 a s jejím vyjádřením pomocí Booleovy algebry 2.1. Existuje však více možností, jak tuto logickou funkci popsat Booleovou algebrou, ale v zájmu minimalizace nákladů při výrobě elektronických zařízení je žádoucí, abychom získali ten nejúspornější zápis funkce, čímž bychom minimalizovali počet hradel potřebných k realizaci

a	b
0	1
1	0



Obrázek 2.3: Hradlo NOT a jeho pravdivostní tabulka



Obrázek 2.4: Dekodér funkce

funkce. Takový zápis funkce se nazývá minimální tvar logické funkce a lze jej získat buď QmC algoritmem, nebo Karnaughovými mapami, přičemž oba způsoby mají stejný výsledek. Proces hledání minimálního tvaru logické funkce se nazývá minimalizace logické funkce. Minimální tvar funkce 2.1 je

$$f(\boldsymbol{v}) = \bar{b}. \tag{2.3}$$

Tento zápis je z hlediska výstupních hodnot ekvivalentní zápisu 2.1 a dekodér funkce bychom zkonstruovali pouhým prohlášením negované vstupní proměnné b za výstup obvodu.

2.5.1 Quine-McCluskey algoritmus

Aplikováním QmC algoritmu na logickou funkci získáme minimální tvar této logické funkce buď v úplné normální disjunktní formě, nebo v úplné normální konjunktní formě. QmC algoritmus je nejvíce využíván pro minimalizaci logických funkcí počítačovými programy, neboť se jedná o přesně definovaný a jednoznačný postup.

Algoritmus se vykonává tak, že nejprve pod sebe vypíšeme soubor všech mintermů. Ve vypsaném souboru budeme hledat dvojice mintermů, jejichž členy se liší právě v jedné proměnné. Na základě každé této dvojice vytvoříme nový implikant, který bude mít stejný tvar jako původní mintermy dvojice, vyloučíme ale rozdílovou proměnnou. Je potřeba nalézt všechny možné dvojice.

Po sestavení všech možných dvojic a vytvoření nových implikantů stejný postup aplikujeme na nově vzniklé implikanty. Implikant či minterm, pro který dvojici už nalézt nemůžeme, je prostý implikant a označíme jej. Pokud nám při procesu vytváření nových implikantů vzniknou 2 shodné, jeden škrtneme a pokračujeme dál pouze s 1. Tento postup opakujeme dokud všechny implikanty nebudou prosté. [7]

Po nalezení všech prostých implikantů vytvoříme tabulku pokrytí. Cílem tabulky pokrytí je vybrat minimum prostých implikantů pro pokrytí všech logických 1 na výstupu funkce. Tabulku sestavíme tak, že do sloupečků napíšeme indexy, kdy funkce nabývá log. 1, a do řádků nalezené prosté implikanty. Následně zjistíme výšku pokrytí jednotlivých indexů. Pokud je výška pokrytí právě 1, musí být prostý implikant vybrán. Díky těmto vybraným implikantům si můžeme vyškrtnout jimy již pokryté indexy. Následně je potřeba optimálně vybrat zbylé implikanty tak, abychom pokryli všechny indexy a zároveň získali minimální tvar funkce. Logickým součtem vybraných implikantů získáváme zápis logické funkce v disjunktní formě.

Výše zmíněný postup si demonstrujme na příkladu. Máme logickou funkci zadanou tabulkou 2.5 a budeme hledat minimální tvar této logické funkce v disjunktní formě.

Tabulka 2.5: Logická funkce 4 vstupních proměnných.

N	a	b	c	d	f
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

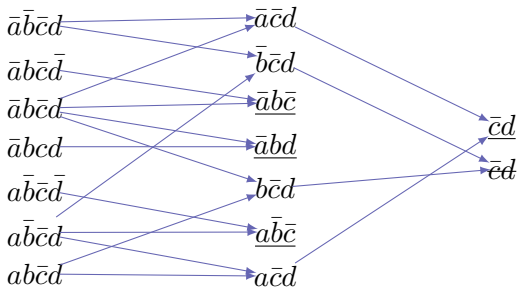
Nejprve je potřeba vypsat soubor mintermů. V tomto případě to bude

Tabulka 2.6: Tabulka pokrytí.

implikanty/indexy	1	4	5	7	8	9	13
~~$\bar{c}d$~~	X		X			X	X
~~$\bar{a}b\bar{c}$~~		X	X				
~~$\bar{a}bd$~~			X	X			
~~$a\bar{b}\bar{c}$~~					X	X	
Výška pokrytí	1	1	3	1	1	2	1

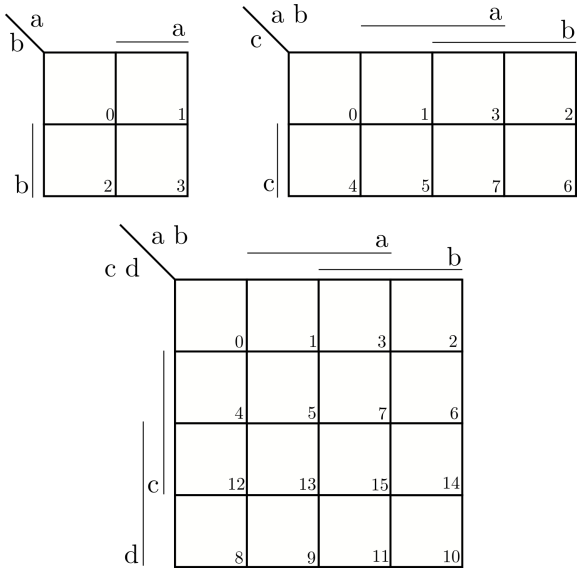
$\bar{a}\bar{b}\bar{c}d$
$\bar{a}b\bar{c}\bar{d}$
$\bar{a}b\bar{c}d$
$\bar{a}bcd$
$a\bar{b}\bar{c}\bar{d}$
$a\bar{b}\bar{c}d$
$ab\bar{c}d$.

Dále je potřeba nalézt prosté implikanty. Proces hledání prostých implikantů je znázorněn na obrázku 2.5. Na levé straně obrázku jsou vypsány



Obrázek 2.5: Grafické znázornění hledání prostých implikantů

mintermy logické funkce. Ke každému mintermu jsme nalezli jiný minterm, který se liší právě v jedné proměnné. Vznikají nám tak implikanty, které jsou vypsány ve druhém sloupečku. Zde se proces hledání dvojic opakuje, k implikantům $\bar{a}b\bar{c}$,$\bar{a}b\bar{d}$, $a\bar{c}d$ však již dvojici nenalézáme, tyto implikanty jsou prosté a podtrhneme je. Další prostý implikant, který nacházíme, je $\bar{c}d$. Tím jsme nalezli všechny prosté implikanty. Nyní vytvoříme tabulku pokrytí 2.6. Pomocí tabuky pokrytí zjistíme, jaké prosté implikanty musíme nutně použít pro zápis funkce. Jsou to ty implikanty, které pokrývají indexy s výškou pokrytí 1. V našem příkladě to jsou shodou okolností všechny implikanty. Pokud by ale po vybrání všech těchto implikantů zůstal nějaký index nepokrytý, bylo by potřeba vybrat optimální kombinaci zbylých prostých implikantů tak, aby byly pokryty všechny indexy. Všechny vybrané implikanty následně



Obrázek 2.6: Karnaughovy mapy pro 2, 3, a 4 proměnné

spojíme logickým součtem. Tím získáváme minimální tvar logické funkce 2.5.

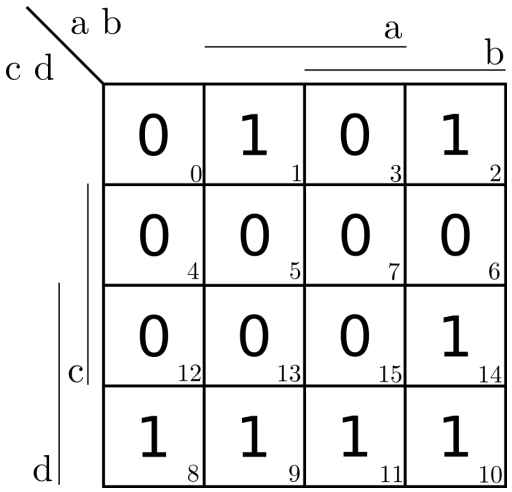
$$f = \bar{c}d + \bar{a}b\bar{c} + \bar{a}bd + a\bar{b}\bar{c}. \tag{2.4}$$

Hledáme-li minimální konjunktní tvar logické funkce, je potřeba na samém začátku minimalizace celou logickou funkci znegovat. S touto negovanou funkcí budeme následně zcela standardně postupovat, tedy nalezneme mintermy, z nich prosté implikanty a z nich minimální disjunktní tvar logické funkce. Následně ale celý výsledek znegujeme. Aplikací De Morganových zákonů získáme minimální konjunktní tvar logické funkce.

2.5.2 Karnaughovy mapy

Minimalizaci logických funkcí do 5 proměnných lze relativně přehledně provést použitím Karnaughových map. Tento postup je vhodný pro minimalizaci logických funkcí při nedostupnosti výpočetních zařízení schopných implementace QmC algoritmu.

Základem tohoto postupu je sestrojení tabulky (tzv. mapy), do které na příslušné pozice vypíšeme odpovídající výstupní hodnoty funkce. Pro funkce s různým počtem vstupních proměnných existují různé mapy. Na obrázku 2.6 jsou Karnaughovy mapy pro 2, 3, a 4 proměnné. Pokud bychom minimalizovali funkci o 4 vstupních proměnných, vyplníme Karnaugovu mapu pro 4 proměnné podle výstupních hodnot funkce pro příslušné indexy. Index každého boxu je uveden v jeho pravém dolním rohu. Zásadní vlastností Karnaughových map je, že se sousední pole liší vždy pouze v 1 proměnné. Logická úroveň všech proměnných pro každou pozici lze zjistit na okrajích mapy => v případě, že je pozice označena linkou dané proměnné, je na této pozici logická proměnná ve stavu logické 1.



Obrázek 2.7: Karnaughova mapa pro 4 proměnné

Po vyplnění mapy výstupními hodnotami minimalizované funkce následuje v procesu minimalizace fáze vytváření smyček obsahujících všechny logické 1 sousedních pozic. Smyčky mohou být pouze velikosti mocnin 2^N, přičemž velikostí smyčky rozumíme počet obsažených logických 1. Pozice spolu sousedí tehdy, když se jím příslušné vstupní logické proměnné liší pouze v 1 proměnné. Podstatné je, abychom nalezli co největší smyčky a aby každá logická 1 v mapě byla obsažena minimálně v 1 smyčce. Smyčky se mohou překrývat. Každou smyčku poté zapíšeme součinem těch logických proměnných, které se v rámci jedné smyčky nemění. Výslednou logickou funkci získáme součtem těchto součinů. Tím získáme minimální disjunktní tvar logické funkce. [8], [9]

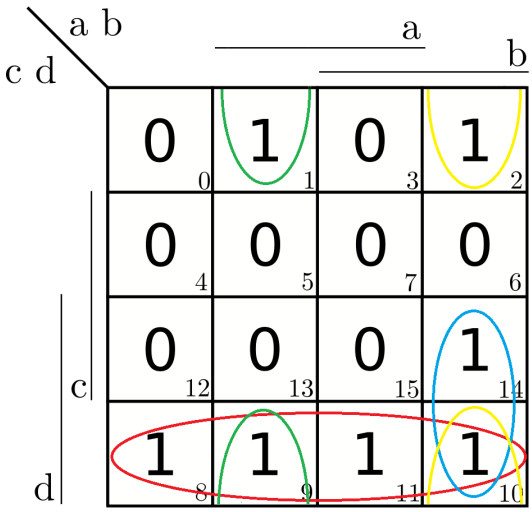
Demonstrujme nyní princip minimalizace logických funkcí pomocí Karnaughových map na příkladu 2.5. V prvním kroku vyplníme Karnaughovu mapu pro 4 proměnné výstupními daty z tabulky, tak jako na obrázku 2.7. Takto vyplněná mapa je připravena pro hledání smyček pokrývající logické 1. Snažíme se nalézt co největší smyčky tak, abychom všechny logické 1 pokryly co nejmenším počtem smyček.

Obrázek 2.8 ukazuje nejlepší možnou konfiguraci. Pro pokrytí všech logických 1 bylo třeba využít 4 smyček, přičemž největší z nich pokryla 4 stavy vstupu funkce. Nyní ke každé smyčce nalezneme odpovídající zápis pomocí součinu vstupních proměnných.

Červená smyčka pokrývá 4 stavy vstupu funkce, v rámci této smyčky se tudíž musí měnit 2 vstupní proměnné, a to a a b. Tyto 2 proměnné do součinu popisující červenou smyčku nezařadíme, získáme součin $\bar{c}d$.

Modrá, zelená i žlutá smyčka pokrývají shodně po 2 logických stavech vstupu funkce. Každá z těchto smyček generuje součin 3 vstupních proměnných. Podle modré smyčky zapíšeme logický součin $\bar{a}bd$, podle zelené $a\bar{b}\bar{c}$ a podle žluté $\bar{a}b\bar{c}$. Získali jsme celkem 4 logické součiny. Disjunktní tvar logické funkce získáme logickým součtem získaných členů. Získali jsme zápis ve tvaru

$$f = \bar{c}d + \bar{a}b\bar{c} + \bar{a}bd + a\bar{b}\bar{c}, \tag{2.5}$$



Obrázek 2.8: Optimální rozložení smyček

odpovídá tak tvaru 2.4, který jsme získali minimalizací s využitím QmC algoritmu.

Využitím Karnaughových map lze taktéž získat minimální konjunktní tvar logické funkce. Hodnoty logické funkce zapíšeme do mapy stejně jako při hledání disjunktního tvaru. Ovšem do smyček neuzavíráme logické 1 jako v předchozím příkladě, nýbrž logické 0. Následně nalezneme smyčkám odpovídající logické součiny, ze součinů vytvoříme disjunktní tvar funkce a ten následně celý znegujeme. Aplikací DeMorganových zákonů získáme minimální tvar logické funkce v konjunktním tvaru. [10]

2.6 Webové stránky

Vzrůstající rychlost a dostupnost internetového připojení zapříčinila, že se velké množství počítačových programů přesouvá do prostředí webových stránek. Výhodami tohoto řešení je možná distribuce výpočetního výkonu na stranu serverů a dále multiplatformnost, jelikož jakékoliv zařízení s odpovídajícím webovým prohlížečem je schopno webové nástroje spustit.

Důležitou součást systému webových stránek tvoří webové prohlížeče. Prohlížeč na základě pokynů uživatele tvoří požadavky skrze HTTP či HTTPS protokol na servery, na kterých jsou dotazované webové stránky umístěny. Každá webová stránka má svůj zdrojový kód, který definuje její chování. Server při bezchybném průběhu pošle zpět zařízení, které poslalo požadavek na webovou stránku, zdrojový kód požadované stránky. Webový prohlížeč tuto odpověď přijme a na základě přijatého zdrojového souboru sestaví a vykreslí webovou stránku. Prohlížeče tedy představují nepostradatelnou komponentu pro uživatele webových stránek.

Zdrojový kód webové stránky je možné sestrojit v jazyce HTML. Na základě zdrojového kódu v HTML webový prohlížeč rozmisťuje elementy grafického

rozhraní webové stránky. Samotné HTML je schopno vytvářet pouze statické webové stránky. To znamená, že existuje pouze jedna verze stránky definovaná ve zdrojovém kódu a není možné měnit či generovat obsah stránek na základě vstupů zadaných uživatelem.

Webovou stránku lze sestrojit také kombinací HTML a vhodného skriptovacího jazyka, nejčastěji PHP nebo JavaScript. V takovém případě hovoříme o dynamické webové stránce. Skriptovací jazyk je schopen zpracovat data zadaná uživatelem do vstupních elementů přítomných na stránce. Na základě těchto dat může skriptovací jazyk generovat obsah zdrojového souboru webové stránky a tím dynamicky reagovat na zadaná uživatelská data.

Rozdíl mezi PHP a JavaScriptem je zásadní. Program PHP upravuje zdrojový kód webové stránky přímo na serveru. Uživatel si prostřednictvím svého prohlížeče stáhne pouze statickou stránku, kde však může do příslušných grafických prvků zadat uživatelská data. Po odeslání zadaných dat prostřednictvím HTTPS protokolu server tyto data přijme, na základě PHP programu zpracuje a případně odpovídajícím způsobem upraví zdrojový kód stránky. Program v JavaScriptu si uživatel do svého počítače stahuje a uživatelův prohlížeč tento program sám implementuje. Výhoda JavaScriptu spočívá v distribuci výpočetní zátěže na pracovní stanice uživatelů. Nevýhodu přístupu JavaScriptu představuje to, že zdrojový kód si uživatel stahuje do svého prohlížeče a je tedy zcela veřejný.

Kapitola 3
Popis řešení

Pro realizaci práce byla zvolena kombinace značkovacího jazyka HTML a programovacího jazyka JavaScript. Výhoda tohoto řešení spočívá v malých nárocích kladených na webový server, a to jak po stránce požadovaného výpočetního výkonu, tak potřebného programového vybavení. Uživateli stačí pro použití nástroje zařízení s jakýmkoliv operačním systémem, které ovšem disponuje internetovým prohlížečem s podporou JavaScriptu.

Vývoj webové aplikace obnáší vytvoření jak grafického rozhraní nástroje, tak i implementaci samotného jádra nástroje, které na pozadí zpracuje data zadaná uživatelem, aplikuje na ně QmC algoritmus a výstup předá grafickému rozhraní.

Vyvíjená webová aplikace se skládá ze 3 souborů. Soubor index.html je základní soubor, který webový prohlížeč načte poté co uživatel pošle požadavek na zobrazení webové aplikace. Dalším souborem je kalkulacka.js. Tento soubor obsahuje veškerou výpočetní logiku nástroje naprogramovanou v jazyce JavaScript. Na tento soubor je uveden odkaz v hlavičce souboru index.html. Posledním souborem je styles.css. Zde je uveden způsob formátování tabulek v aplikaci. V principu by moha aplikace fungovat i bez tohoto souboru, grafické rozhraní by ovšem nepůsobilo přehledně. Veškeré úryvky kódu, které budou v následujících sekcích popisovány, jsou vyňaty ze souboru kalkulacka.js.

3.1 Grafické rozhraní

Základním požadavkem na grafické rozhraní je jeho přehlednost. Uživatel by měl být schopen se ihned po otevření nástroje zorientovat a být chopen ho okamžitě používat. Toho dosáhneme využitím jednoduchého a jednoznačného grafického vzhledu. Obrázek 3.1 ukazuje grafické rozhraní nástroje po jeho spuštění. Uživatel se v této fázi rozhoduje, jakým způsobem nástroji zadá logickou funkci. V případě, že si uživatel vybere buď ”Zadání výpisem hodnot”, nebo ”Náhodný příklad”, bude následovat výběr počtu vstupních proměnných minimalizované logické funkce pomocí radiobuttonů. Po výběru počtu vstupních proměnných v případě náhodného příkladu následuje pravdivostní tabulka s náhodnými výstupními hodnotami, v případě konkrétního zadání následuje pravdivostní tabulka vyplněná nulami a přepokládá se, že uživatel tabulku vyplní svými hodnotami. Stisknutím tlačítka ”Zjednoduš”

Obrázek 3.1: Rozhraní nástroje po jeho spuštění

zadaná data načte jádro nástroje, které na ně aplikuje QmC algoritmus a výstupem bude minimální disjunktní a minimální konjunktní tvar logické funkce. Minimální tvary se následně vypíší pod zadanou pravdivostní tabulku.

3.2 Jádro programu

Jádrem programu je myšlena část programu, se kterou uživatel nepřichází do styku, ale přesto je pro funkci webové aplikace nezbytná, neboť právě tato část programu na uživatelem zadaných datech vykonává QmC algoritmus a generuje minimální tvar vložené funkce. Jádro jsme naprogramovali v programocacím jazyce JavaScript s využitím rozhraní DOM.

3.2.1 Spuštění aplikace

Jakmile uživatel spustí webovou aplikaci (načte webovou stránku na svém prohlížečí), JavaScript skrze event onload a využitím rozhraní DOM vytvoří počáteční strukturu stránky. To zahrnuje i vytvoření 3 tlačítek pro volbu zadání logické funkce. K těmto tlačítkům je přidán EventListener, který po stisknutí tlačítka vykoná příslušnou akci, tedy buď možnost nahrání vstupního souboru, nebo vygeneruje radiobuttony pro výběr počtu vstupních proměnných pravdivostní tabulky.

3.2.2 Načítání zadaných hodnot

Načítání zadaných hodnot se liší, pokud načítáme hodnoty ze souboru, nebo z pravdivostní tabulky. Zatímco při načítání hodnot z tabulky je potřeba nejprve definovat počet vstupních proměnných a poté zjistit logické úrovně pro každý stav vstupu funkce. V případě načítání funkce ze souboru je potřeba v zadaném souboru dekódovat indexy a vyfiltrovat oddělovací znaky.

Při načítání hodnot z pravdivostní tabulky tedy vytvoříme podle zvoleného počtu vstupních proměnných pravdivostní tabulku, jak ukazuje obrázek 3.2.

```
for (var i = 0; i < rad.length; i++) {
   rad[i].addEventListener('change',function(){
     createtable(this,false);
   }
  );
}
```

Obrázek 3.2: Vytvoření tabulky pravdivostních hodnot

```
let but=document.createElement("button");
but.innerHTML="zjednoduš!";
area[0].appendChild(but);
but.addEventListener('click',function(){
    calculate(loadoutputs());
}
);
```

Obrázek 3.3: Inicializace tlačítka ”Zjednoduš!”

Kód ke každému radiobuttonu přidá EventListener, který v případě změny stavu daného buttonu zavolá funkci *createtable*. Tato funkce vytvoří celou pravdivostní tabulku a zobrazí ji uživateli. První parametr funkce je instance příslušného radiobuttonu. Druhým argumentem volíme, zda-li chceme výstup funkce vyplnit náhodnými logickými hodnotami.

Jakmile funkce tabulku vytvoří a zobrazí, je potřeba inicializovat tlačítko ”Zjednoduš!”, po jehož stisknutí nástroj začne minimalizovat zadanou funkci. Na obrázku 3.3 je ukázka kódu, který na první řádce vytvoří tlačítko, dále jej popíše, zobrazí na grafickém rozhraní a nakonec nastaví event handler po kliknutí na tlačítko na zavolání funkce *calculate*, která jako parametr vyžaduje pole vstupních hodnot. Toto pole vstupních hodnot získáme funkcí loadoutputs().

Pokud se uživatel rozhodne, že logickou funkci do aplikace zadá pomocí txt souboru, není potřeba zobrazovat ani výběr počtu proměnných, ani tabulku pravdivostních hodnot. Uživatel do webové aplikace pouze nahraje soubor s indexy, na kterých je výstupní hodnota funkce 1. Nahrání souboru do webové aplikace je realizováno prvkem HTML ”input” typu ”file”, jak je ukázáno na obrázku 3.4.

Nahraný soubor je potřeba dekódovat. Část kódu 3.5 se vykoná po zadání pokynu ke zjednodušení. První 3 řádky inicializují načtení obsahu nahraného txt souboru. Jakmile je obsah souboru načten, spustí se obsluha eventu onload, ve které se prostřednictvím funkce *decodefile()* převede řetězec obsažený v txt souboru na pole polí logických hodnot. Jednotlivá zadání se pak jedno po druhém řeší ve funkci *calculate()*.

3.2.3 QmC algoritmus

Načtením vstupních hodnot jsme si vytvořili vstupní podmínky pro QmC algoritmus. Máme tedy pole hodnot, které obsahuje buď 0, nebo 1, nebo X pro

```
<input type="file" id="file-selector" accept=".txt">
```

Obrázek 3.4: Prvek input pro vložení textového souboru

```
let fileSelector = document.getElementById('file-selector');
let fr=new FileReader();
fr.readAsText(fileSelector.files[0]);
fr.onload=function(){
  let ex=decodefile(fr.result);
  for(let y=0;y<ex.length;y++){
    calculate(ex[y]);
  }
}
```

Obrázek 3.5: Dekódování textového souboru

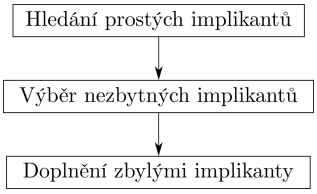
neurčitý stav. Hodnoty mají v poli stejné indexy, jako mají indexy u logické funkce. Získání minimálního tvaru logické funkce pomocí QmC algoritmu je rozděleno do 3 fází podle obrázku 3.6.

V první fázi je důležité nalézt a programově definovat mintermy. V této fázi budeme ke všem X označujícím neurčité stavy přistupovat jako k logickým 1. Minterm je v našem programu reprezentován polem délky 2, přičemž první pozice představuje hodnotu indexu, kterému minterm přísluší a na druhé pozici je maska, která má logickou 1 na tom binárním řádu, který odpovídá řádu logické proměnné, která byla z příslušného implikantu vyřazena. Graficky je na příkladu proces vytvoření pole mintermů znázorněn na obrázku 3.7.

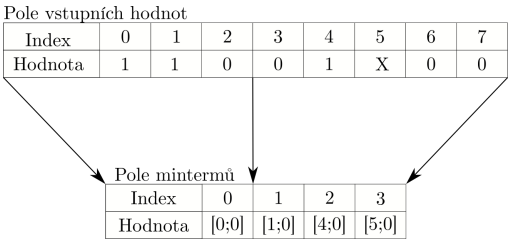
Pro další výklad bude praktické si definovat systém odkazování na prvky v poli mintermů/implikantů. Odkaz na prvek budeme značit dvojicí dvojic hranatých závorek s číslami, tedy $[u][v]$, kde u je index pole mintermů a v je index vloženého pole. Například [2][0] odkazuje na hodnotu 4 podle obrázku 3.7. Indexování startuje od 0.

Nalezené implikanty využijeme při hledání dvojic, jehož algoritmus je na obrázku 3.8. Postupujeme tak, že si vezmeme 2 impliknaty z pole implikantů a poté zjistíme, jestli obsahují stejné proměnné tím, že porovnáme jejich masky. Ty se musejí shodovat. Dále pokračujeme vykonáváním bitových logických součtů hodnoty $[v][0]$ a binárních čísel, která obsahují pouze 1 cifru rovnu logické 1 a všechny ostatní cifry jsou logické 0. Tím získáme tolik výsledků, kolik má logická funkce proměnných. Tyto výsledky reprezentují implikanty, které by mohly tvořit dvojici s implikantem $[v]$, neboť se liší právě v 1 proměnné. Musíme ale zjistit, zda-li tento potenciální člen dvojice vůbec existuje v poli mintermů. To zjistíme podmínkou na 5. řádku obrázku 3.8. Pokud je splněna, člen se kterým může implikant $[v]$ utvořit dvojici, byl nalezen, my jej zaznamenáme do pole implikantů a smažeme původní 2 implikanty, ze kterých byla dvojice sestavena. Tento postup provedeme pro všechny dvojice z pole implikantů, čímž získáme pole nových implikantů. Graficky bychom na obrázku 2.5 získali druhý sloupeček.

Pro nalezení obecnějších implikantů zopakujeme postup popsaný v předchozím odstavci s tím rozdílem, že za vstupní implikanty dosadíme pole



Obrázek 3.6: Proces minimalizace logické funkce



Obrázek 3.7: Získávání mintermů

nalezených implikantů z předchozí iterace. Postup opakujeme do doby, kdy nejsme schopni nalézat nové dvojice implikantů. V této fázi jsme nalezli prosté implikanty.

Výpočetní náročnost této části algoritmu je ovlivněna počtem logických 1 na výstupu funkce. Protože musíme vytvořit a otestovat všechny možné dvojice nalezených implikantů, je náročnost této fáze dána kombinačním číslem $\binom{N}{2}$, kde N je počet implikantů. V první iteraci hledání implikantů je N rovno počtu logických 1 na výstupu funkce.

Nyní se na obrázku 3.6 přesouváme do fáze výběru nezbytných implikantů. V této fázi využijeme hodnoty načtené z grafického rozhraní a z pole prostých implikantů. Všechna X, která označují neurčité stavy v poli vstupních hodnot, v této fázi převedeme na logické 0. Připravíme si prázdné pole nezbytných implikantů *results*. Nejprve postupujeme, jak ukazuje obrázek 3.10. Hledáme logické 1 v poli vstupních hodnot *input[i]*, kde i postupně nabývá hodnot indexů funkce. Pokud ji nalezneme, musíme zkontrolovat, zda-li již tato logická 1 není pokryta některým z nezbytných implikantů nalezených dříve. Zkontrolujeme to tak, že provedeme bitovou negaci testovaného prvku $[y][1]$ z pole nezbytných implikantů a s tímto dále provedeme bitový logický součin s hodnotou indexu pole vstupních hodnot, na kterém jsme nalezli danou logickou 1. Jestliže se tento výsledek shoduje s hodnotou $[y][0]$ z pole nezbytných implikantů, je index tímto nezbytným implikantem pokryt a pokračujeme na další index. Pokud ovšem shoda neplatí, pokryta tímto implikantem není. V takovém případě musíme dále zjistit, zda-li tato logická 1 z pole vstupních hodnot má výšku pokrytí rovnu 1. To zjistíme kódem na obrázku 3.9. Pole *matrix3* obsahuje prosté implikanty, proměnná i představuje indexy funkce nabývající logické 1 a pole *results* agreguje nezbytné implikanty. Podmínka na 3. řádku testuje, zda-li je index i pokryt prostým implikantem *matrix3[y]*. Pokud ano, tento implikant je zaznamenán do pole *results* a nastavíme příznak *onematch* signalizující, že daný index byl pokryt právě jedním implikantem.

```
for(let v=0;v<out.length;v++){
  for(let i=1;i<out.length-v;i++){
    if(out[v][1]==out[v+i][1]){
      for(let y=0b000000000001;y<0b100000000;y=y<<1){
        if((out[v][0]|y)==out[v+i][0]&&out[v][0]!=out[v+i][0]){
          out.push([out[v][0]&(~y),out[v][1]|y]);
          dlt.push(v);
          dlt.push(v+i);
        }
      }
    }
  }
}
```

Obrázek 3.8: Hledání dvojic implikantů

```
onematch=false;
for(let y=0;y<matrix3.length;y++){
  if((i&(~matrix3[y][1]))==matrix3[y][0]&&!onematch){
    onematch=true;
    results.push(matrix3[y]);
  }
  else if((i&(~matrix3[y][1]))==matrix3[y][0]&&onematch){
    results.pop();
    break;
  }
}
```

Obrázek 3.9: Výběr nezbytných implikantů

Podmínka na 7. řádku je splněna tehdy, když je index *i* pokryt více než jedním implikantem. V tomto případě je nutné vyjmout z pole *results* nezbytný implikant, který byl do pole nahrán při nastavování příznaku *onematch* do hodnoty true. Jakmile proměnná *i* postupně nabyde všech indexů funkce odpovídajících logickým 1, je pole *results* naplněno nezbytnými implikanty a 2. fáze procesu minimalizace logické funkce končí.

Poslední fází je nalezení doplňujících implikantů pro plné pokrytí všech indexů logické funkce. Opět využijeme hodnoty načtené z grafického rozhraní a z pole prostých implikantů, navíc použijeme i pole nezbytných implikantů, ve kterém budeme uchovávat implikanty potřebné k sestavení logické funkce. Všechna X, která označují neurčité stavy v poli vstupních hodnot, zde převedeme na logické 0. V postupu na obrázku 3.11 pole *input* obsahuje výstupní hodnoty minimalizované funkce, pole *implicants* obsahuje všechny nalezené prosté implikanty a pole *nesimplicants* uchovává nezbytné implikanty nalezené v předchozím postupu. Podmínka ve smyčce *while* prostřednictvím funkceu *nboundinputs()* zjistí, jestli ve funkci existují nepokryté indexy. Pokud ano, vykoná se tělo smyčky, ve kterém se do pole *nesimplicants* přidá prostřednictvím funkce *coverindex()* ten prostý implikant, který pokrývá nejvíce doposud nepokrytých indexů. Smyčka while zajistí opakování tohoto procesu, dokud

```
for(let i=0b0;i<input.length;i++){
  onematch=false;
  nomatch=true;
  if(input[i]==1){
    for(let y=0;y<results.length;y++){
      if((i&(~results[y][1]))==results[y][0]){
        nomatch=false;
        break;
      }
    }
```

Obrázek 3.10: Testování pokrytí indexu nezbytnými implikanty

```
function coverunboundunputs(input,implicants,nesimplicants){
  while(unboundinputs(input,nesimplicants).length!=0){
    nesimplicants.push(coverindex(unboundinputs(input,nesimplicants),implicants));
  }
  return nesimplicants;
}
```

Obrázek 3.11: Doplnění zbylými implikanty

nebudou všechny indexy logické funkce pokryty. Tímto zajistíme, že se výsledek minimalizace bude skládat z minimálního počtu implikantů. Tělo funkce *coverindex()* je na obrázku 3.12. Pole *indexes* obsahuje dosud nepokryté indexy funkce, pole *implicants* obsahuje všechny prosté implikanty. Do pole *freq* se ukládá počet pokrytých indexů implikanty na příslušných indexech. Funkce *coverindex()* nakonec vrátí implikant z pole *implicants*, jehož index je roven indexu maximální hodnoty v poli *freq*.

Tímto jsme dokončili poslední část získání implikantů pro minimální tvar logické funkce v disjunktním tvaru. Všechny potřebné implikanty jsou uložené v poli. Pokud bychom chtěli získat minimální konjunktní tvar logické funkce, stačí pouze před celým procesem naznačeným na obrázku 3.6 výstupní hodnoty logické funkce negovat. Pokud je uživatelem zadaná výstupní hodnota logická 1, změníme ji na logickou 0 a naopak. Neurčitý stav X zůstává neurčitým stavem. V následující podkapitole zmíníme odlišný způsob dekódování minimálního disjunktního tvaru a minimálního konjunktního tvaru.

3.2.4 Dekódování a převedení implikantů na grafický výstup

Předchozí výklad popsal způsob získání pole, ve kterém jsou systematicky zaznamenány prosté implikanty minimálního tvaru logické funkce. Nyní je potřeba tyto implikanty dekódovat a transformovat do textové podoby za účelem zobrazení výsledku minimalizace uživateli. Dekódování se liší pro minimální disjunktní a konjunktní tvar.

Obrázek 3.13 popisuje získání textového řetězce v minimálním disjunktním tvaru. Pole *res* obsahuje prosté implikanty minimálního tvaru logické funkce a proměnná *varcnt* je rovna počtu vstupních proměnných minimalizované

```
function coverindex(indexes,implicants){
  let freq=[];
  for(let i=0;i<indexes.length;i++){
    for(let y=0;y<implicants.length;y++){
      if((indexes[i]&(~implicants[y][1]))==implicants[y][0]){
        freq[y]++;
      }
    }
  }
  return implicants[indexOfMax(freq)];
}
```

Obrázek 3.12: Funkce coverindex

```
let str="";
for(let i=0;i<res.length;i++){
  for(let y=0;y<varcnt;y++){
    if(((res[i][1]>>>y)&0b1)==0){
      if(((res[i][0]>>>y)&0b1)==1){
        str+=letters[y];
      }
      else{
        str+=negletters[y];
      }
    }
  }
  if(i+1!=res.length){
    str+="+";
  }
}
```

Obrázek 3.13: Vytvoření disjunktního řetězce z pole prostých implikantů

funkce. Pole *letters* a *negletters* obsahují textové znaky reprezentující přímé a negované vstupní proměnné. Pro každý implikant z pole *res* postupně prozkoumáváme, jaké proměné jsou v daném implikantu obsaženy. Je-li proměnná v implikantu obsažena, zapíšeme do řetězce *str* tuto proměnnou buď v přímé podobě prostřednictvím pole *letters*, nebo v negované z pole *negletters*. Mezi jednotlivé implikanty v řetězci *str* vkládáme znak '+'. Výstupem kódu na obrázku 3.13 je textový řetězec, který je následně zobrazen uživateli jako minimální disjunktní tvar logické funkce. Tímto jsme dokončili celý proces minimalizace logické funkce pomocí QmC algoritmu.

Minimální konjunktní tvar logické funkce je získán podobně, mezi proměnné jednoho implikantu však vkládáme logický součet '+', a mezi jednotlivé implikanty logický součin.

```
for(let x=0;x<input.length;x++){
  let high=[];
  for(let y=0;y<highlight.length;y++){
    if(isincluded(x,highlight[y])){
      high.push(y+1);
    }
  }
  if(input[x]==1){
    map[kmaps[Math.log2(input.length)-1][x][0]][kmaps[Math.log2(input.length)-1][x][1]]=[1,high];
  }
  else if(input[x]==0){
    map[kmaps[Math.log2(input.length)-1][x][0]][kmaps[Math.log2(input.length)-1][x][1]]=[0,high];
  }
  else{
    map[kmaps[Math.log2(input.length)-1][x][0]][kmaps[Math.log2(input.length)-1][x][1]]=['X',high];
  }
}
```

Obrázek 3.14: Vytvoření 2D pole s Karnaughovou mapou a pokrytím smyčkami

3.3 Karnaughova mapa

Karnaughovy mapy mají hlavní využití v rámci minimalizace logických funkcí při nedostupnosti výpočetní techniky. Relativně rychle a snadno lze takto minimalizovat logické funkce do 5 vstupních proměnných. Pro minimalizaci logických funkcí s využitím výpočetní techniky se využívá převážně QmC algoritmu. Karnaughovy mapy představují grafický postup přehledný pro člověka. Mapu minimalizované logické funkce v této práci reverzně sestavujeme z již hotového minimálního disjunktního tvaru logické funkce získaného QmC algoritmem.

Kód na obrázku 3.13 vytvoří 2D pole *map* reprezentující Karnaughovu mapu a nahraje na příslušné pozice odpovídající logické hodnoty a k nim i smyčky, kterými je daná pozice v mapě pokryta. Pole *input* obsahuje výstupní hodnoty minimalizované funkce, pole *highlight* obsahuje prosté implikanty minimálního tvaru funkce, podle kterých budeme Karnaughovu mapu vytvářet. Smyčka na 3. řádku společně s podmínkou na 4. řádku testuje, kterými implikanty je každý index pokryt. Tyto implikanty zaznamenáváme do pole *high*. V další fázi plníme pole *map*, přičemž odkazování v tomto poli přesně odpovídá pozicím v Karnaughově mapě. Například *map[x][y]* odkazuje na x pozici v horizontálním směru mapy a na y pozici ve vertikálním směru mapy. Po získání pole *map* stačí toto pole vypsat do tabulky v grafickém rozhraní a vybarvit příslušné smyčky.

3.4 Souborová struktura, umístění na webový server

Výhodou webové aplikace je její dostupnost prostřednictvím webového prohlížeče. Uživatel si nemusí do svého zařízení stahovat žádný spustitelný soubor,

Obrázek 3.15: Výběr počtu vstupních proměnných logické funkce

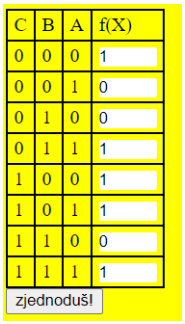
navíc se jedná o multiplatformní řešení, tudíž je webová aplikace, jejíž běhové prostředí zajišťuje samotný webový prohlížeč, spustitelná na jakémkoliv moderním prohlížeči.

Webhosting naší webové aplikace jsme si zřídili na hostingové službě Endora od společnosti Webglobe s.r.o. Zároveň s tím jsme si zaregistrovali doménu *www.qmcsolver.cz*, na které bude naše webová aplikace dostupná. Webovou aplikaci na server webhostingové služby nahrajeme prostřednictvím webového rozhraní. Do kořenového adresáře nahrajeme soubor index.html. Tento soubor je načten uživatelem po zadání požadavku na načtení domény *www.qmcsolver.cz*. Dále do kořenového adresáře umístíme soubor styles.css, který upravuje vzhled tabulek webové aplikace, a složku js. Do složky js vložíme soubor kalkulacka.js. V tomto souboru je obsažena výpočetní část webové aplikace. Tím jsme umístili nástroj na webový server a je nyní dostupný na doméně *www.qmcsolver.cz*.

3.5 Příklad použití nástroje

Uživateli vznikl požadavek na minimalizaci logické funkce 3 proměnných. Funkci má uživatel zadánu pravdivostní tabulkou.

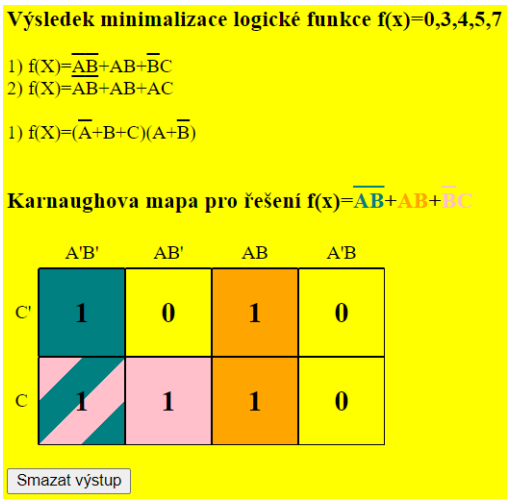
Otevřením webového prohlížeče a zadáním adresy *www.qmcsolver.cz* načteme požadovanou webovou aplikaci, čímž získáme rozhraní představené na obrázku 3.1. Pokračujeme kliknutím na tlačítko ”zadání výpisem hodnot” a následným vybráním počtu vstupních proměnných prostřednictvím radiobuttonů 3.15. Jakmile počet proměnných vybereme, nástroj vygeneruje odpovídající pravdivostní tabulku s textovými poli určenými k zapsání výstupních hodnot funkce 3.16. Po zadání výstupních hodnot logické funkce a stisknutím tlačítka ”Zjednoduš” jsou zadané hodnoty načteny jádrem programu a dále



C	B	A	f(X)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

zjednodušl

Obrázek 3.16: Pravdivostní tabulka pro zadání logické funkce



Výsledek minimalizace logické funkce f(x)=0,3,4,5,7

1) f(X)=$\overline{AB}$+AB+$\overline{B}$C
2) f(X)=$\overline{AB}$+AB+AC

1) f(X)=($\overline{A}$+B+C)(A+$\overline{B}$)

Karnaughova mapa pro řešení f(x)=$\overline{AB}$+AB+$\overline{B}$C

	A'B'	AB'	AB	A'B
C'	1	0	1	0
C	1	1	1	0

Smazat výstup

Obrázek 3.17: Výstup minimalizace

jsou použity jako vstupní data pro QmC algoritmus. Po zpracování těchto dat je výsledek vypsán na grafické rozhraní v minimální disjunktní a minimální konjunktní formě. Zároveň je pro disjunktní formu vykreslena odpovídající Karnaughova mapa. Výsledek minimalizace je na obrázku 3.17.

Kapitola 4

Závěr

Zadáním této práce bylo vytvořit online nástroj, který minimalizuje zadanou logickou funkci. Tento nástroj by měl obsahovat přehledné uživatelské rozhraní, do kterého uživatel zadá logickou funkci, kterou chce zjednodušit.

Nástroj byl implementován v jazyce HTML s využitím programovacího jazyka JavaScript, tudíž je možné jej využít skrze všechny moderní webové prohlížeče. Nástroj umožňuje minimalizaci logických funkcí až 6 vstupních proměnných.

Grafické rozhraní, které jsme zde implementovali, je velmi stručné a přímočaré. Každý uživatel by měl být schopen se ihned zorientovat a nástroj umět použít. Zadání logické funkce je rovněž velmi ergonomické. Stačí pouze rozkliknout textové pole náležící nejnižšímu indexu, zadat výstupní hodnotu logické funkce a stisknutím klávesy Tabulátor se přesunout na následující výstupní hodnotu. Minimální tvar logické funkce se poté vypisuje v úplné normální disjunktní formě i v úplné normální konjunktní formě ve více možných variantách.

Jádro programu, které se stará o validaci hodnot zadaných uživatelem a o samotnou minimalizaci, bylo implementováno v JavaScriptu. Díky tomu je výpočet minimálního tvaru funkce distribuován na zařízení koncových uživatelů a není tím zatěžován server. Samotný algoritmus se podařilo naprogramovat pouze s využitím relativně jednoduchých instrukcí, převážně bitových operací, které jsou nenáročné na vykonávání počítačem. Celý online nástroj má velikost zhruba 10 kB. Načtení nástroje prohlížečem by tedy mělo být, při běžně dostupné rychlosti připojení k internetu, velmi rychlé.

Možným rozšířením této práce je vytvořit přehledné uživatelské rozhraní pro mobilní telefony. Nástroj je možné použít v současném stavu na mobilních zařízeních, rozhraní je však navrženo na obrazovky osobních počítačů. Dále by užitečnou funkci představoval generátor schématu zapojení logických obvodů implementujících zadanou logickou funkci s odkazem na stránky prodejce příslušných obvodů.

Literatura

[1] Binary number. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 12 Květen 2021 [cit. 2021-5-17]. Dostupné z: https://en.wikipedia.org/wiki/Binary_number.

[2] Poziční číselná soustava. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-17]. Dostupné z: https://cs.wikipedia.org/wiki/Pozi%C4%8Dn%C3%AD_%C4%8D%C3%ADseln%C3%A1_soustava.

[3] Boolean algebra. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-17]. Dostupné z: https://en.wikipedia.org/wiki/Boolean_algebra.

[4] Boolean Algebra. Blavatnik school of computer science [online]. Tel Aviv: Blavatnik school of computer science, 2008 [cit. 2021-5-17]. Dostupné z: https://www.cs.tau.ac.il/ nin/Courses/ComStruct04/BooleanAlgebra.htm.

[5] Úvod do logiky: Úplná disjunktivní / konjunktivní normální forma a její minimalizace [online]. Plzeň: Západočeská univerzita v Plzni, 2014 [cit. 2021-5-17]. Dostupné z: https://www.esf.kfi.zcu.cz/logika/opory/phk1102/VL_09_UDNF.pdf.

[6] LAFATA, Pavel. Kombinační vs. sekvenční logické obvody. Praha, 2019. Dostupné také z: https://moodle.fel.cvut.cz/pluginfile.php/220006/mod_resource/content/1/04_Kombina%C4%8Dn%C

[7] Minimalization of Switching Functions using Quine-McCluskey Method. International Journal of Computer Applications [online]. 2013, 2013(4), 12-16 [cit. 2021-5-17]. Dostupné z: doi:http://dx.doi.org/10.5120/14103-2127.

[8] LAFATA, Pavel, Michal PRAVDA a Petr HAMPL. Digitální technika. Praha: České vysoké učení technické, 2011. ISBN 8001049140.

[9] LAFATA, Pavel. Minimalizace logických funkcí a jejich realizace pomocí hradel: Přednáška předmětu B2B32DITA. Praha, 2019. Dostupné také z: https://moodle.fel.cvut.cz/pluginfile.php/220005/mod_resource/content/1/03_Logick%C3%A9_funk

[10] ANTOŠOVÁ Marcela, V. D.: *Číslicová technika*. České Budějovice: KOPP, 2009, ISBN 978-80-7232-394-4, 308 s.

Kapitola 5

Příloha

Příloha obsahuje zdrojový kód webové aplikace, bakalářskou práci, obrázky použité v práci a soubor se vzorovým zadáním 3 logických funkcí, který je možné nahrát do online nástroje.