

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH VÝPOČETNÍCH STRUKTUR V CELULÁRNÍCH AUTOMATECH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JINDŘICH LUŽA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH VÝPOČETNÍCH STRUKTUR V CELULÁRNÍCH AUTOMATECH

DESIGN OF COMPUTING STRUCTURES IN CELLULAR AUTOMATA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

JINDŘICH LUŽA

VEDOUcí PRÁCE

SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2014

Abstrakt

Cílem této práce je prozkoumat možnosti realizace výpočetních struktur za pomoci celulárních automatů. Práce popisuje principy fungování celulárních automatů a zabývá se jejich způsoby jejich využití pro stanovený cíl. Na 1D a 2D rozměrných celulárních automatech vybraných typů jsou ukázány možné způsoby jak Turingovsky univerzálního výpočtu tak i další řešící specifické úlohy. Tímto je demonstrována schopnost celulárních automatů provádět výpočet a zároveň jsou ukázány rozličné způsoby interpretace vstupů a výstupů výpočtů na celulárním automatu. S přihlédnutím k těmto poznatkům jsou pro vybrané obvody navrženy testy mající za úkol nalézt realizaci těchto obvodů na celulárních automatech za pomoci zvoleného evolučního algoritmu. Nalezené výsledky jsou pak porovnány z hlediska jejich nároků na evoluční algoritmus a spotřebované výpočetní zdroje.

Abstract

The goal of this master thesis is to examine possibilities of realizing computational structures in cellular automata. The work describes the fundamental principles of cellular automata and summarizes some ways of how to achieve the specified goal. An overview of Turing-complete and other specialized computational tasks is proposed considering both 1D and 2D cellular automata. It is shown that different computational scenarios in cellular automata can be considered with various setups of the input and output arrangements. With regard to showed inputs and outputs arrangement, sets of tests is designed to find solutions of choosen computational structures on cellular automata with use of choosen evolutionary algorithm. Found solutions are compared by computational resources consumption and difficulty of discovery later.

Klíčová slova

Celulární automaty, univerzální výpočet, uniformní celulární automaty, neuniformní celulární automaty, Game of Life, evoluční algoritmy.

Keywords

cellular automata, universal computation, uniform cellular automata, non-uniform cellular automata, Game of Life, evolutionary algorithms.

Citace

Jindřich Luža: Návrh výpočetních struktur v celulárních automatech, diplomová práce, Brno, FIT VUT v Brně, 2014

Návrh výpočetních struktur v celulárních automatech

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jindřich Luža
26. května 2014

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Michalu Bidlovi, Ph.D. za odborné vedení a konzultace, které mi pro vypracování tohoto projektu poskytl.

© Jindřich Luža, 2014.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1	Úvod	4
2	Celulární automaty	6
2.1	Úvod do celulárních automatů	7
2.2	Základní celulární automat	9
2.3	Neuniformní celulární automaty	10
2.4	Podmínková pravidla	11
3	Univerzalita celulárních automatů a počítání na celulárních automatech	13
3.1	Turingův stroj	13
3.2	Univerzální výpočet na 1D celulárním automatu	14
3.3	Univerzální výpočet na automatu Game of Life	14
3.4	Univerzální výpočet na 2D neuniformním celulárním automatu	17
3.5	Další možnosti počítání na celulárních automatech	18
3.5.1	Počítání na 1D celulárních automatech	18
3.5.2	Binární počítání na 2D celulárních automatech	18
3.5.3	Specializované počítání na celulárních automatech	19
4	Evoluční algoritmy	21
4.1	Princip evolučních algoritmů	21
4.2	Genetický algoritmus při konstrukci celulárního automatu	24
4.2.1	Genetický algoritmus při hledání nesystematického řešení celulárních automatů	25
5	Návrh výpočtů v CA pomocí hledání nesystematického řešení	29
5.1	Sčítání v binárním 2D CA	29
5.2	Násobení v binárním 2D CA	31
5.3	N^2 ve vícestavovém 1D CA	33
6	Experimentální výsledky	35
6.1	Sčítání 2 bitových operandů	35
6.1.1	Příklady nalezených řešení	40
6.2	Sčítání 3 bitových operandů	41
6.2.1	Příklady nalezených řešení	43
6.3	Násobení 2 bitových operandů	44
6.3.1	Příklady nalezených řešení	49
6.4	N^2 ve vícestavovém 1D CA	50
6.5	Mobilita nalezených řešení	51
6.6	Shrnutí dosažených výsledků	52

7	Závěr	53
A	Manuál pro kolekci testovacích skriptů	57
A.1	Obsah CD	57
A.2	Požadavky na běh testů	57
A.3	Kompilace	57
A.4	Spuštění experimentu	58
A.5	Popis konfiguračního souboru experimentu	58
A.6	Vygenerování průběhu řešení	59
A.7	Vygenerování statistik pro výsledky běhů	60

Seznam obrázků

2.1	Von Neumannovo a Moorovo okolí	7
2.2	Výpočet nového stavu buňky	7
2.3	Ukazka konfigurací Game of life	8
2.4	Průběh pravidla 30 [20]	10
2.5	Vyhodnocení podmínkového pravidla	11
3.1	Přechodové pravidla 1D CA simulující univerzální Turingův stroj	14
3.2	Vývoj gosperova glider gunu	15
3.3	Logické hradla konstruované v celulárním automatu Game of Life	16
3.4	Dělič frekvence generování kluzáků	16
3.5	Reflektor kluzáků a eater objekt	17
3.6	Tabulka pravidel pro neuniformní univerzální celulární automat	18
3.7	Pravidlo 132 jako dektor lichých čísel. Prostřední buňka v posledním kroku určuje zda je číslo liché nebo sudé	18
3.8	1D automat počítající mocninu čísla	19
3.9	binární operace na 2d celulárních automatech	19
3.10	Příklad zakódování vstupů a výstupů do CA pro nalezení nesystematického řešení	20
4.1	Celkový pohled na vyhodnocování a průběh GA	28
5.1	Volba polohy buněk pro vstupy a výstup CA - sčítání. $C_3C_2C_1 = A_2A_1 + B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.	30
5.2	Volba polohy buněk pro vstupy a výstup CA - sčítání 3 bitových operandů. $C_4C_3C_2C_1 = A_3A_2A_1 + B_3B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.	31
5.3	Volba polohy buněk pro vstupy a výstup CA - násobení. $C_3C_2C_1 = A_2A_1 * B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.	32
5.4	Upravené varianty pro násobení 3 bitových operandů	33

6.1	Rozložení počtu generací potřebných k nalezení řešení	37
6.2	Rozložení počtu kroků ke splnění podmínek	38
6.3	Rozložení počtu pravidel použitých řešením	39
6.4	Příklad sčítání 2 bitových operandů - varianta A - řešení o 5 krocích + jeden krok navíc	39
6.5	Příklad sčítání 2 bitových operandů pro vybrané podmínky - varianta D na rozšířené ploše automatu o 2 + v kroku 4 o 3 (viz. sekce 6.5)	40
6.6	Příklad sčítání 2 bitových operandů - varianta E - řešení o 6 krocích	41
6.7	Rozložení počtu generací potřebných k nalezení řešení	42
6.8	Rozložení počtu kroků ke splnění podmínek	42
6.9	Rozložení počtu pravidel použitých řešením	42
6.10	Příklad sčítání 3 bitových operandů pro vybrané podmínky - varianta D - první 4 kroky	43
6.11	Příklad sčítání 3 bitových operandů pro vybrané podmínky - varianta E - první 4 kroky	44
6.12	Rozložení počtu generací potřebných k nalezení řešení	46
6.13	Rozložení počtu kroků ke splnění podmínek	47
6.14	Rozložení počtu pravidel použitých řešením	48
6.15	Příklad násobení 2 bitových operandů - varianta D - řešení o 2 krocích + 1 krok	48
6.16	Příklad násobení 2 bitových operandů pro vybrané podmínky - varianta C na rozšířené ploše automatu o 1 (viz. sekce 6.5)	49
6.17	Příklad násobení 2 bitových operandů - varianta E - řešení o 3 krocích . . .	49
6.18	Statistické grafy pro druhou mocninu	50
6.19	Příklad průběhu mocniny	50

Seznam tabulek

2.1	Wolframův celulární automat - pravidlo 30	9
4.1	Pravdivostní tabulky zvolených operátorů	28
5.1	Výsledky 100 běhů pro nalezení přechodové funkce pro sčítání dvoubitových operandů	30
5.2	Výsledky 100 běhů pro nalezení přechodové funkce pro sčítání tříbitových operandů	31
5.3	Výsledky 100 běhů pro nalezení přechodové funkce pro násobení dvoubitových operandů	32
5.4	Výsledky 100 běhů pro nalezení přechodové funkce realizující druhou mocninu	34
6.1	Přechodová tabulka pravidel pro příklad na obr. 6.4	36
6.2	Přechodová tabulka pravidel pro příklad na obr. 6.15	45
6.3	Tabulka zvětšení prostoru CA	51

Kapitola 1

Úvod

Celulární automaty jsou jako výpočetní model vhodným prostředkem pro řešení celé řady reálných problémů pocházejících z různých oblastí vědy. Mezi hlavní přednosti CA, které byly naznačeny výše, patří zejména masivní paralelismus, ve kterém můžeme každé buňce automatu přiřadit výpočetní prostředek a nový globální stav pak spočítat v jednom kroku. Dále pak přítomnost pouze lokální interakce buněk a neexistence globálního řadiče, homogenní struktura, velmi jednoduchá pravidla lokálních interakcí, při kterých lze dosáhnout velmi složitěho globálního chování [18]. Tyto vlastnosti dělají z celulárních automatů vhodný model pro implementaci v hardwaru. Takto implementovaný výpočetní model by nebyl díky homogenní struktuře finančně nákladný, vykazoval by odolnost proti chybám - chybnou buňku lze narozdíl od specializovaného výpočetního obvodu jednoduše nahradit či případně navrhnout automat tak, aby byl schopný se částečně vyrovnat s chybami v některých buňkách - což je možné díky absenci globálního řízení a spolehání se pouze na lokální interakce. Při vhodné implementaci je rovněž schopen rekonfigurace pro vykonávání jiné činnosti. Tato schopnost spolu s aplikací evolučních algoritmů je využita ve výzkumu evolwaru (evolučního hardwaru).

Možnost využití celulárního automatu jako výpočetního nástroje prozkoumal už von Neumann, když dokázal výpočetní úplnost svého univerzálního konstruktora [3]. Stejně tak bylo dokázána výpočetní univerzalita Conwayova automatu Game of Life (viz. sekce 3.3) a Langtonova mravence (důkaz lze najít v [5]). Fungováním tradičních logických obvodů a realizací univerzálního výpočetního modelu v 2D neuniformním automatu se zabýval Sipper [18]. Univerzalitu celulárního automatu v jedné dimenzi se zabýval A.R. Smith a vytvořil univerzální 1D celulární automat s poloměrem okolí $r=1$ a 29 stavů [11].

Cílem této práce je na vybraných výpočetních obvodech prozkoumat možnost jejich realizace na celulárních automatech za pomoci zvoleného evolučního algoritmu s vhodnými parametry a následně porovnat případné nalezená řešení z hlediska jejich náročnosti na výpočetní zdroje a náročnosti jejich nalezení.

V úvodu druhé kapitoly je popsán teoretický model celulárního automatu, jsou zde vysvětleny pojmy, které jsou v práci použity, a na příkladu je znázorněno, jak celulární automat pracuje. Následuje historický úvod do problematiky celulárních automatů nastiňující hlavní důvody jejich výzkumu. Po popisu základního celulárního automatu a demonstraci jeho vývoje jsou okrajově zmíněny neuniformní celulární automaty. Kapitola je završena popisem podmínkových pravidel přechodové funkce, která jsou v této práci použita.

Třetí kapitola se zabývá univerzalitou celulárních automatů a možnostmi, jak pomocí nich lze počítat. V jejím úvodu je stručně popsáno, co znamená univerzalita v doméně celulárních automatů s krátkým popisem Turingova stroje, ke kterému se vztahuje. Násle-

dují detailní příklady možností univerzálního počítání na celulárních automatech. Kapitola vrcholí představením dalších způsobů, kterými lze realizovat výpočet na celulárních automatech.

Čtvrtá kapitola pojednává o evolučních algoritmech. V úvodu je popsán Thomasonův experiment, známý pro použití právě evolučních algoritmů. Následuje obecný popis evolučních algoritmů s vysvětlením používaného názvosloví a popisem zákonitostí, které je třeba brát na zřetel. Úvod zakončuje krátký popis nejznámějších evolučních algoritmů. Dále je popsáno, jak lze aplikovat genetický algoritmus, který je pro tuto práci zvolen jako evoluční algoritmus, na vývoj celulárních automatů. Na konkrétním způsobu aplikace genetického algoritmu jsou popsány vlivy nastavení jeho parametrů a způsoby jeho vyhodnocování. Přesný průběh a vyhodnocování genetického algoritmu s popisem hodnot zvolených pro sadu experimentu završuje čtvrtou kapitolu.

Zvolené obvody pro realizaci jsou popsány v kapitole páté. Po nastínění obecných podmínek, které musí realizace obvodů splňovat, následuje popis realizací konkrétních obvodů s obrazovou dokumentací variant volby umístění vstupních a výstupních operandů obvodu. U každého zvoleného obvodu je pak přiložena tabulka se souhrnými výsledky z běhů experimentů.

Šestá kapitola zachycuje detailní výsledky běhů experimentů s detailními statistikami pro jednotlivé varianty obvodů. U každého obvodu je pro jednotlivé varianty porovnávána úspěšnost experimentů, náročnost na nalezení řešení a jeho složitost. Následují příklady nalezených řešení s jejich detailním průběhem a přechodovou funkcí.

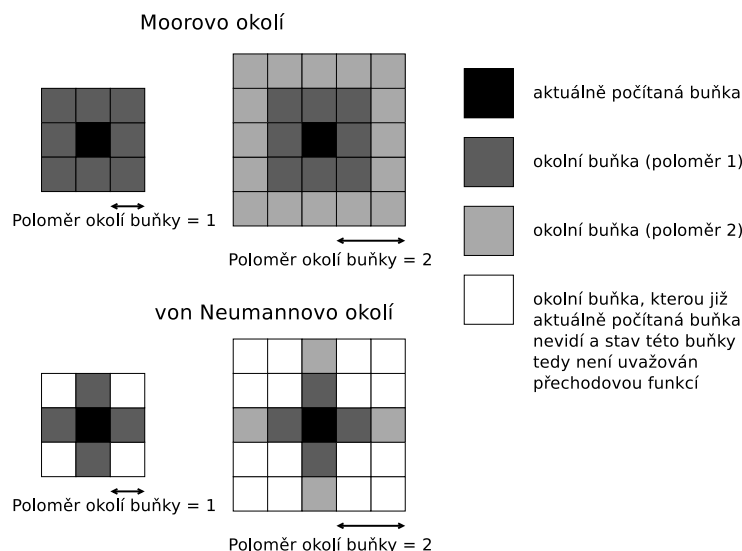
Práce navazuje na semestrální projekt, v rámci něhož byla vypracována teoretická část končící sekcí 4.2.

Kapitola 2

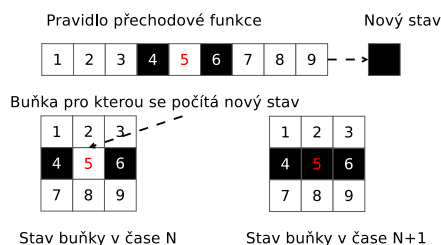
Celulární automaty

Celulární automat je systém buněk synchronně měnící svůj stav v diskrétním čase na základě aktuálního stavu každé buňky a aktuálního stavu buněk v jejím okolí podle přechodové funkce. Nejčastěji je systém buněk celulárního automatu chápán jako 1 až 3 dimenzionální mřížka. Ta může být rozměrově nekonečná nebo vymezena maximálním počtem buněk v každém směru. Při omezeném množství buněk se často využívá pravidlo cyklických krajových podmínek, což znamená, že první a poslední buňka se shodnými rozměrovými souřadnicemi (například stejný řádek v 2D mřížce) jsou sousedící [16]. Při pevných okrajových podmínkách je mřížka větší v každém rozměru o $2r$ buněk, kde r označuje poloměr okolí. Napevno určené okrajové buňky nejsou vyhodnocovány přechodovou funkcí a počas celého vývoje automatu nemění svůj stav. Okolím buňky je chápána buňka samotná a její nejbližší sousední buňky v celulárním automatu. Poloměr okolí je celočíselná maximální vzdálenost sousední buňky, která ještě spadá pod okolí, od buňky středové, pro kterou je okolí definováno. Okolí s poloměrem 1 bude tedy mimo středovou buňku obsahovat buňky, které mají alespoň jednu ze souřadnic o 1 větší nebo menší než středová buňka. Nejčastější typy okolí buňky ve 2D mřížce jsou 5-okolí, které obsahuje buňku samotnou a přímo sousedící buňky ve svislém a vodorovném směru, a 9-okolí, které oproti 5-okolí obsahuje navíc buňky po uhlopříčkách od středové buňky (viz. obr. 2.1). 5-okolí se často nazývá von Neumannovo a 9-okolí Moorovo. Stav buňky je obvykle chápán jako celé číslo. Množina stavů, které může buňka nabývat, je konečná. Každá buňka se nachází právě v jednom stavu z konečné množiny stavů. Každá buňka CA může obvykle nabývat libovolné hodnoty z množiny stavů. Pokud toto pravidlo neplatí, jedná se o neuniformní celulární automat (viz sekce 2.3).

Matice stavů všech buněk CA v určitém čase je nazývána konfigurace CA. Přejchod z konfigurace CA v čase t do konfigurace v čase $t + 1$ je označen jako krok automatu. Přejchod buňky ze stavu v čase t do stavu v čase $t + 1$ je vyobrazen na obr. 2.2. Buňka na základě stavu svého a svého okolí změní svůj stav podle přechodové funkce definované pro daný CA. Přejchodová funkce je společná pro všechny buňky automatu. Pokud toto neplatí, jedná se o neuniformní celulární automat (viz. sekce 2.3). Přejchodová funkce automatu musí být úplně definována, aby bylo možné jednoznačně určit nový stav buňky. Velmi často se přechodová funkce zapisuje formou tabulky, přičemž pro kombinace, které v tabulce nejsou uvedeny, se předpokládá, že výsledný nový stav buňky bude shodný se současným stavem. Ještě více úsporným zápisem přechodové funkce je použití podmínkových pravidel viz sekce 2.4.



Obrázek 2.1: Von Neumannovo a Moorovo okolí



Obrázek 2.2: Výpočet nového stavu buňky

2.1 Úvod do celulárních automatů

Koncept celulárního automatu vymysleli S. Ulam a J. von Neumann koncem padesátých let 20. století jako formální nástroj ke studiu reprodukcí se organismů [3]. Původní von Neumannův automat pracoval ve dvou rozměrné síti buněk s propojením každé buňky se čtyřmi sousedními buňkami ve horizontálním a vertikálním směru. Von Neumann následně navrhl konkrétní celulární automat schopný sebereplikace za pomoci konstrukčního ramena. Jeho automat, zvaný také Von Neumannův univerzální konstruktor, sestával z 29 stavů odhadem zabíral asi 50 000 až 200 000 buněk [6]. Princip automatu spočíval v schopnosti konstrukce potomka podle instrukcí na pásce, která byla součástí rodiče. Bylo dokázáno že von Neumannův univerzální konstruktor je výpočetně úplný [3].

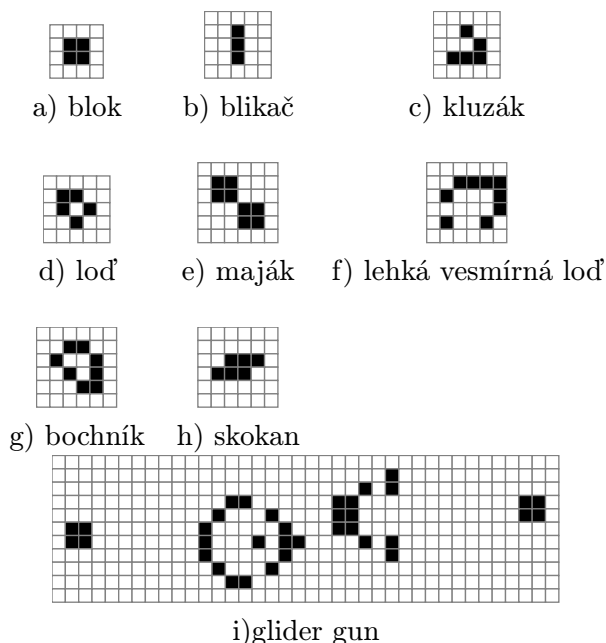
Edgar F. Codd rozšířil von Neumannovu práci a dokázal, že k vytvoření stroje schopného sebereplikace postačuje 8 stavů [6]. Skutečná implementace von Neumannova a Coddova automatu proběhla mnohem později kvůli limitovaným možnostem tehdejší výpočetní techniky. Christopher Langton ještě více zjednodušil Coddův automat tím, že se vzdal konstrukční univerzálnosti a sestavil automat zvaný Langtonova smyčka, který pro svou replikaci potřeboval pouhých 86 buněk a 8 stavů [6]. Později vzniklo mnoho dalších variací Langtonovy smyčky.

Zájem o výzkum celulárních automatů rapidně vzrostl díky J. H. Conwayovi. Conway se zajímal o von Neumannovu práci a vytvořil automat vykazující komplexní chování, schop-

nost replikace a evoluce. Conwayova verze automatu byla oproti von Neumanově verzi značně zjednodušená. Conway namísto čtyř okolních buněk využil všech 8 okolních buněk a pouhé 2 možné stavy buňky, které označovaly zda je buňka živá nebo mrtvá. Průběh výpočtu automatu se řídil podle čtyř jednoduchých pravidel:

1. živá buňka, která má méně než 2 živé sousedy, umírá v důsledku osamocení.
2. živá buňka, která má 2 nebo 3 živé sousedy, přežívá do další generace.
3. živá buňka, která má více než 3 živé sousedy, umírá v důsledku vyhladovění.
4. mrtvá buňka, která má přesně 3 živé sousedy, ožívá jako důsledek reprodukce.

Conway svůj automat nazval Hra života (Game of Life). Ten vzbudil velký zájem odborné veřejnosti pravděpodobně z důvodu neočekávaných výsledků vývoje konfigurace automatu v čase při volbě různých počátečních nastavení [18]. Typy vývoje lze rozdělit na stálé, oscilující nebo nad meze rostoucí. Konfigurace stálého automatu je časově neměnná. Oscilující automat se po určité periodě opět navrátí do původní podoby - původního vzoru seskupení živých buněk, přičemž toto nemusí být vždy na stejné pozici jako v předchozí periodě, ale může “putovat” v prostoru automatu. Nad meze rostoucí automat vykazuje tendenci neustále zvyšovat počet živých buněk. Nejjednodušší příklady stálých konfigurací automatu jsou blok (a), bočník (g) a loď (d). Z oscilujících automatů které zachovávají pozici vzoru na ploše jsou to blikáč (c), skokan (h) a máják (e), z oscilujících konfigurací putující po ploše jsou to potom kluzák (c) a odlehčená vesmírná loď (f). Nejznámější nad meze rostoucí konfigurace automatu je gosperův glider gun (i) (viz obr. 2.3).



Obrázek 2.3: Ukazka konfigurací Game of live

Další překvapivý výsledek složitého vývoje konfigurace celulárního automatu poskytuje Langtonův mravenec. Langtonův mravenec je celulární automat s ještě jednoduššími pravidly než Game of Life. Princip spočívá v umístění mravence na plochu automatu, jejíž buňky

mohou být černé a nebo bílé barvy. Mravenec se pohybuje po ploše automatu a buňka, na kterou vstoupil v následujícím kroku změni svou barvu. Pohyb mravence je řízen dvěma pravidly. Pokud mravenec vstoupil na buňku bílé barvy, v následujícím kroku vstoupí na buňku ležící 90° vpravo od současně obsazené buňky. Pokud mravenec vstoupil na buňku černé barvy následující krok povede na buňku ležící 90° vlevo od současně obsazené buňky [9].

Chování, které vykazuje Langtonův mravenec, je prvních pár set kroků velmi jednoduché. Mravenec generuje po ploše automatu jednoduché vzory, které jsou často symetrické. Po prvních pár stech krocích se začne mravenec po ploše pohybovat tak, že generuje pseudonáhodnou cestu po ploše. Po asi 10 000 krocích začne mravenec generovat na ploše pravidelné opakující se vzory v nekonečném cyklu. Podrobný popis lze najít v [10]. Nejzajímavější na Langtonově mravenci je tvrzení, že výsledné chování mravence - generování pravidelných vzoru v nekonečném cyklu - není ovlivněno počáteční konfigurací.

2.2 Základní celulární automat

Základní celulární automat nebo také Wolframův celulární automat je jednorozměrný uniformní celulární automat o dvou stavech a poloměrem okolí $r = 1$ v nekonečném prostoru. Stav buňky a stav okolí buňky může nabývat 2^3 konfigurací. Spolu v kombinaci s novým stavem v dalším kroku je celkový počet možných pravidel přechodové funkce základního celulárního automatu roven 256 (2^{2^3}). Například Wolframův celulární automat pravidla 30 tedy obsahuje přechodovou funkci vyjádřenou podle tabulky 2.1

Wolfram zkoumal celulární automaty a na základě jejich chování v čase s různými počátečními konfiguracemi je rozdělil do 4 tříd [18]:

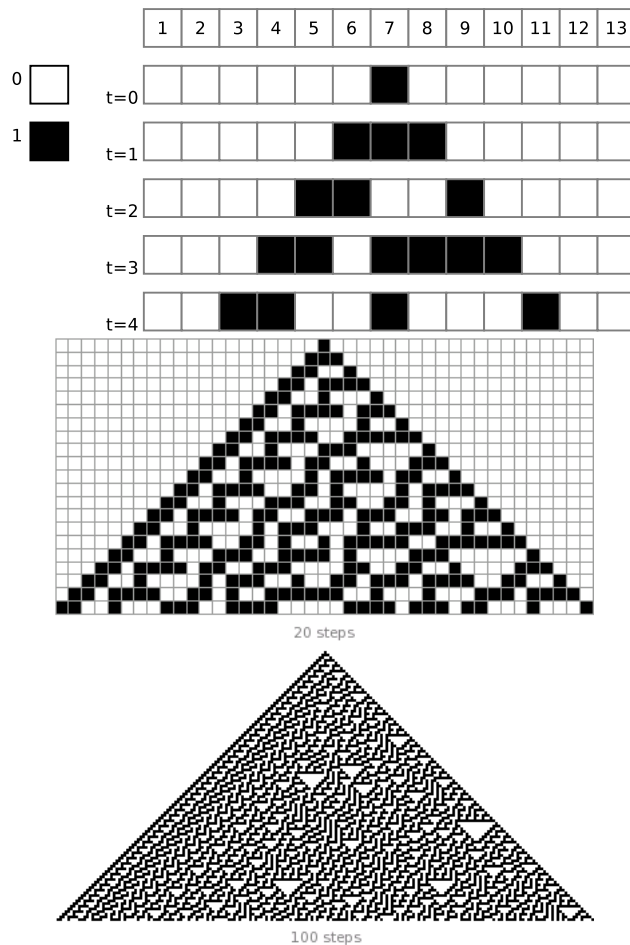
1. třída I s homogenním chováním stavů - konfigurace automatu v čase bude konečná
2. třída II s periodickým opakováním stavů
3. třída III s chaotickým chováním stavů - viz. pravidlo 30
4. třída IV s komplexním chováním lokalizovaných struktur - chování na hranici mezi třídou 2. a 3.

Průběh přechodů konfigurací automatu pravidla 30 je vyobrazen na obr.2.4. Počáteční konfigurace automatu v čase 0 obsahuje pouze jednu buňku ve stavu 1 a ostatní ve stavu 0. Buňka na pozici 6 má vpravo souseda ve stavu 1 a proto se na ni aplikuje aplikuje pátý

i	1	2	3	4	5	6	7	8
$c^t - 1$	0	1	0	1	0	1	0	1
c^t	0	0	1	1	0	0	1	1
$c^t + 1$	0	0	0	0	1	1	1	1
c^{t+1}	0	1	1	1	1	0	0	0

Tabulka 2.1: Wolframův celulární automat - pravidlo 30

Řádek c^t značí aktuální možný stav buňky c v čase t . Řádky $c^t + 1$ a $c^t - 1$ popisují možné stavy pravého a levého souseda buňky c . Řádek c^{t+1} určuje výsledný stav buňky c v čase $t + 1$. Sekvence nových stavů v řádku c^{t+1} převedena do desítkové soustavy zastupuje číslo 30. Proto je tato přechodová funkce nazvána pravidlo 30.



Obrázek 2.4: Průběh pravidla 30 [20]

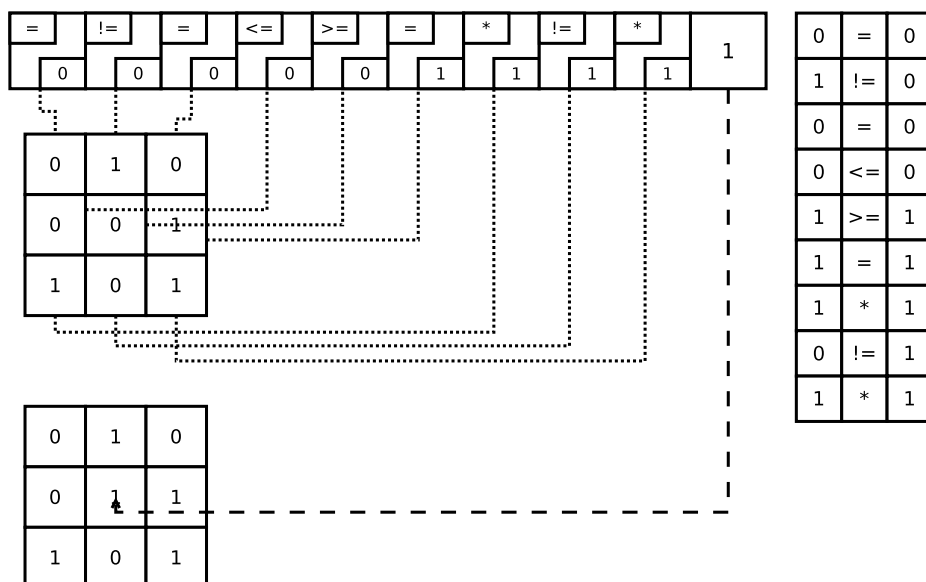
sloupec z přechodové tabulky pravidla 30. Stav buňky 6 v čase 1 bude tedy 1. Buňka 7 má oba sousedy ve stavu 0, proto se na ni aplikuje třetí sloupec z přechodové tabulky a v čase 1 bude mít stav 1. Buňka 8 má levého souseda ve stavu 1. Po uplatnění sloupce 2 z přechodové tabulky bude mít v čase 1 stav 1. V průběhu 20 kroků jsou již patrné náznaky nepravidelností, které jsou pak zřetelně vidět na zobrazení průběhu 100 kroků. Pravidlo 30 patří do třídy III.

2.3 Neuniformní celulární automaty

Neuniformní celulární automaty sdílí všechny žádoucí vlastnosti s uniformními celulárními automaty (např. masivní paralelismus, lokální interakce a synchronní časově diskrétní krok) [18]. Jak již bylo zmíněno v kapitole 2, neuniformní celulární automat umožňuje definovat různé přechodové funkce pro odlišné buňky, případně definovat různé množiny stavů, kterých mohou buňky nabývat. V sekci 3.3 je naznačen způsob univerzálního výpočtu na automatu Game of Life, který je následován popisem univerzálního výpočtu stejným způsobem na neuniformním automatu (viz. sekce 3.4). Úspory na počtu použitých buněk neuniformního řešení jsou při srovnání těchto dvou přístupů jasně patrné.

2.4 Podmínková pravidla

V kapitole 2. jsou zmíněna podmínková pravidla jako prostředek sloužící ke zredukování případného počtu řádků tabulky reprezentující přechodovou funkci CA. Použití podmínkových pravidel nejen redukuje počet řádků potřebných k zápisu přechodové funkce, ale zároveň velmi usnadňuje a zrychluje průběh genetického algoritmu (a obecně i jakéhokoliv jiného evolučního algoritmu použitého při hledání přechodové funkce CA). Základní principem podmínkových pravidel je zredudování více možných konfigurací, ve kterých se může buňka a její okolí vyskytovat, do jednoho pravidla. Podmínkové pravidlo se skládá z logického operátoru podmínky a napevno zvolené konstanty, která slouží jako druhý operand pro podmínkový operátor. Prvním operandem je hodnota buňky, na kterou se podmínka vztahuje. Množina logických operátorů použitých pro podmínková pravidla může být libovolná. Pokud jsou výsledky vyhodnocení operátorů pro buňku a všechny buňky okolí kladné, znamená to, že bylo splněno podmínkové pravidlo, a nový stav buňky bude nastaven podle výstupního stavu podmínkového pravidla. Grafická demonstrace vyhodnocování podmínkového pravidla je vykreslena na obr.2.5.



Obrázek 2.5: Vyhodnocení podmínkového pravidla

Pravidlo obsahuje podmínkový operátor v levém horním rohu a konstantu v pravém dolním. Jako druhý operand operátoru je navázán stav buňky. Přiřazení konkrétní buňky ke konkrétnímu operátoru je naznačeno tečkovanými čarami. Pokud je vyhodnocení všech operátorů (tabulka vlevo) úspěšné, stav buňky bude nastaven na novou hodnotu podle podmínkového pravidla (naznačeno čárkovanou spojnici). Operátor $*$ značí, že vyhodnocení bude vždy pravdivé bez ohledu na hodnotu buňky nebo konstanty. Operátor \neq značí, že stav buňky musí být jiný než konstanta. Operátory \leq a \geq značí menší/větší nebo rovno než konstanta

Vhodná volba operátorů dokáže v jednom pravidle pokrýt několik kombinací stavů buňky a jejího okolí. Tato redukce je markatnější při použití většího počtu možných stavů buňky. Zmíněná výhoda je ale částečně i nevýhodou, a to sice v tom, že není možné rychle určit, zda jedno pravidlo nepokrývá pravidlo druhé. Při použití běžných pravidel přechodové funkce musí všechny stavy okolí buňky přesně odpovídat stavům pravidla. Pokud je tato funkce reprezentována tabulkou, která bude vyhodnocována od shora dolů, dokud nebude

nalezen řádek, který vyhovuje kombinaci stavů okolí buňky, nezáleží na pořadí řádků tabulky, protože vyhovující řádek bude vždy maximálně jeden (za předpokladu, že řádky pro kombinace okolí, ve kterých buňka nemění stav, nejsou uvedeny). Při použití podmínkových pravidel tento fakt neplatí - může se tedy stát, že se pokrytí kombinací stavů okolí buňky bude překrývat. V tom případě se vyhodnotí vždy první pravidlo v tabulce. Tato skutečnost vnáší do hledání přechodové funkce reprezentované podmínkovými pravidly problém zkrácení použitého počtu pravidel. Pokud se například na prvním řádku vyskytne pravidlo se všemi operátory *, které vrací vždy logickou 1, další pravidla se už neprovedou a cíleného chování CA se nemusí dosáhnout. Na druhou stranu může zase pravidlo, které pokrývá některé další vstupní kombinace jiného pravidla, zabránit nechtěné odchylce v chování.

Kapitola 3

Univerzalita celulárních automatů a počítání na celulárních automatech

Při výzkumu celulárních automatů je často věnován nemalý čas k určení zda je celulární automat výpočetně univerzální. Výpočetně univerzální celulární automat je takový automat, který dokáže spočítat libovolnou funkci spočitatelnou výpočetním strojem v dané třídě výpočetních strojů. V případě této práce se pojem výpočetní univerzalita vztahuje k Turingovu stroji. Výpočetně univerzální celulární automat je tedy takový automat, který dokáže efektivně spočítat všechny problémy, které dokáže efektivně spočítat Turingův stroj a naopak ty, které nedokáže spočítat Turingův stroj, nedokáže spočítat ani daný celulární automat.

K určení toho, zda je jiný výpočetní prostředek výpočetně úplný (univerzální) se používá důkaz, že libovolný Turingův stroj, nebo jiný výpočetní stroj pro který již byla univerzalita dokázána, lze simulovat právě tímto výpočetním prostředkem. Ke splnění těchto podmínek je často nutné zobecnit konstrukci Turingova stroje a umožnit počáteční konfiguraci pásky s nekonečně opakujícím se prázdným slovem na pravé nebo obou stranách pásky, případně umožnit počáteční konfiguraci pásky nekonečně periodicky se opakující. Výpočetním modelům splňujícím Turingovskou úplnost za zmírnění těchto podmínek, říkáme Turingovsky slabé nebo poloslabé [14].

3.1 Turingův stroj

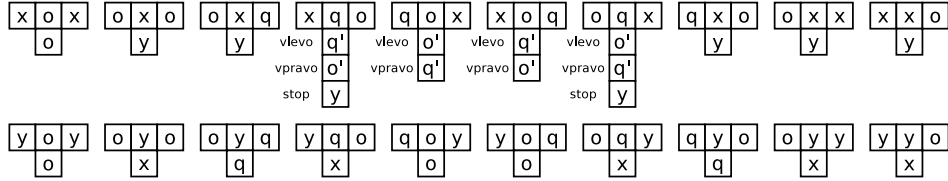
Turingův stroj je matematický model jednoduchého stroje popisující univerzální prostředek pro výpočet jakékoliv spočitatelné funkce. Stroj sestává z jednorozměrné pásky nekonečné v jednom směru, čtecí/přepisovací hlavy a stavového registru. Páska je rozdělena na jednotlivé buňky, které obsahují jeden z konečné množiny symbolů Turingova stroje. V každém výpočetním kroku přečte stroj symbol buňky, na které se aktuálně nachází čtecí/přepisovací hlava. Na základě přečteného symbolu a současného obsahu stavového registru stroj změní svůj stav, přepíše symbol v buňce pod čtecí/přepisovací hlavou a případně posune hlavu o jednu buňku doleva nebo doprava. Výpočet Turingova stroje je korektně ukončen ve chvíli, kdy stroj přejde do akceptujícího nebo neakceptujícího koncového stavu. V případě, že je pro zadaný vstup výpočet konečný, stroj vždy zastaví na akceptujícím nebo neakceptujícím stavu. Pokud pro zadaný vstup výpočet konečný není, cyklí stroj v nekonečné smyčce[17].

3.2 Univerzální výpočet na 1D celulárním automatu

Výpočetní univerzalita 1D celulárního automatu bude ilustrována na schopnosti simulovat univerzální Turingův stroj. Pro simulaci TS níže popsaným způsobem je potřeba, aby počátek konfigurace automatu byla nastavena na periodicky se opakující se vzor pro rozlišení struktury pásky TS.¹Následujícím způsobem lze simulovat libovolný Turingův stroj o m stavech a k symbolech s výsledným počtem $m + k + 2$ celkových stavů konstruovaného celulárního automatu [11]. Pro zobecnění bude použito označení q a σ značící stav a symbol Turingova stroje. Symboly x a y budou pomocné symboly oddělující jednotlivé buňky Turingova stroje. Počáteční konfigurace buněk celulárního automatu bude tedy následující:

$$x \quad q \quad \sigma_0 \quad x \quad \sigma_0 \quad x \quad \sigma_0 \quad x \quad \dots \quad \sigma_\infty \quad x$$

Pravidla přechodové funkce jsou stanovena na obrázku 3.1. Symboly souhrně označené q značí mimo stavu TS i pozici hlavy na pásce. Zastavení simulovaného TS je provedeno přepsáním symbolu q symbolem y



Obrázek 3.1: Přechodové pravidla 1D CA simulující univerzální Turingův stroj

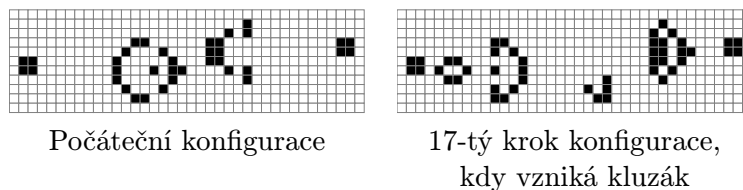
Symboly x a y slouží jako oddělovače buněk Turingova stroje a zároveň jako počítadla poloviny kroku TS. Symbol x se přepíná bez výjimky do symbolu y a obráceně. Pokud obsahuje celulární automat buňky ve stavu x , jedná se o půlkrok, ve kterém proběhne přepis symbolu TS a posun hlavy TS. Pokud jsou oddělovače ve stavu y , jedná se o druhý půlkrok, kdy je hlava TS přesunuta na příslušnou platnou buňku a to za oddělovačem. Jak je patrné z obrázku a zmíněno výše, přechod konfigurace TS může nastat pouze pokud probíhá první půlkrok TS (oddělovače jsou ve stavu x) a to konkrétně ve stavech xqo a oqx . Výsledný stav závisí na přechodovém pravidlu TS a posunu hlavy, jak je naznačeno třemi možnými výslednými stavy buňky (vlevo, vpravo, stop).

Stavy q' a o' indikují nový stav TS a přepsaný symbol pásky, který vznikne přechodem konfigurace $(q, o, n) \rightarrow (q', o', n')$. Stavy q' a o' jsou stejnou generalizací stavů a symbolů TS jako stavy q a o a proto nejsou zobrazeny v možnostech přechodů druhého půlkroku. Přechody v druhém půlkroku TS (oddělovače jsou ve stavu y) vedou pouze k posunutí hlavy.

3.3 Univerzální výpočet na automatu Game of Life

Celulární automat Game of Life i přes jednoduchá přechodová pravidla buněk vykazuje schopnost velmi komplexního chování. Jak bylo popsáno výše, vývoj stavu automatu GoL lze rozdělit na stálý, oscilující a nadmeze rostoucí. Právě nad meze rostoucí vývoj automatu pro počáteční konfigurace se stal klíčovým pro dokázání, že je GoL schopný provádět univerzální výpočet a vůbec pro způsob, jakým lze počítat na 2D celulárním automatu. Jak bylo zmíněno výše, nejznámější nadmeze rostoucí konfigurace Game of Life je Gosperův glider gun. V počáteční konfiguraci se jedná o 2 objekty pozicované tak, že vývoj počáteční konfigurace způsobí jejich kolizi. Výsledkem této kolize je kluzák, který vzniká přesně

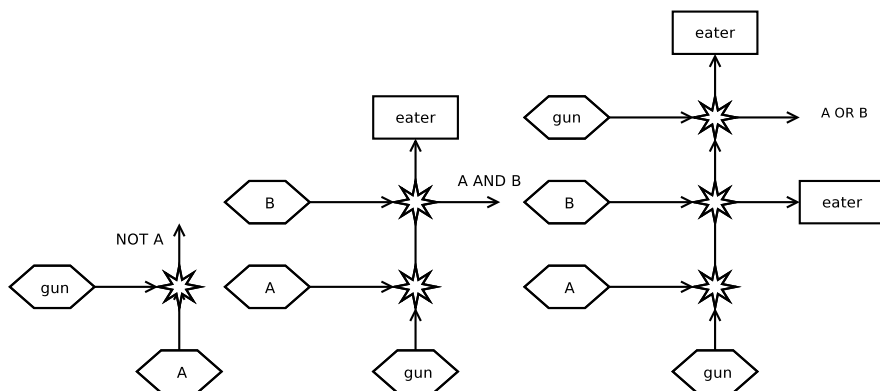
¹Což je podmínka porušující konstrukci Turingova stroje a proto je výsledný automat Turingovsky poloslabý. Neperiodicky se opakující okolí lze simulovat obdobně za použití většího počtu stavů.



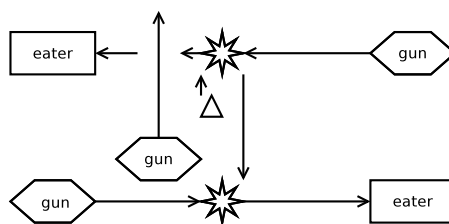
Obrázek 3.2: Vývoj gosperova glider gunu

uprostřed kolizního objektu a putuje diagonálně směrem dolů. Zbytky kolizního objektu putují horizontálně v obou směrech ke čtvercovým blokům umístěným po obou krajích, od kterých se “odrazí” a tímto odrazem nově vzniklé objekty opět putují v kolizním směru proti sobě, až se v určité chvíli automat dostane do konfigurace, která byla jeho počáteční konfigurací. Celková konstrukce tedy umožňuje vysílat nekonečný retěz kluzáků směrem diagonálně vpravo dolů (při použití počáteční konfigurace jako v obr. 3.2). Samotný kluzák pak může kolidovat s dalším objektem ve své cestě. Pokud dva kluzáky putují proti sobě, ve správném nastavení poloh dojde k jejich srážce a výsledkem této srážky je zánik obou kluzáků. Naopak při jiném nastavení poloh jeden z kluzáků zanikne a druhý bude putovat dál, ovšem v obráceném směru. Pokud kluzák koliduje s objektem zvaným eater, což je objekt se stálou konfigurací, kluzák zaniká, ale eater se vrací po kolizi během čtyř kroků zpět do své původní konfigurace. Je nutné poznamenat že kýžený výsledek výše zmíněných kolizí závisí na přesném časování a správně zvolených pozicích.

Nicméně lze takto sestavit automat simulující chování logických obvodů. Gosperův glider gun bude použit pro generování signálu log 1 s danou periodou. Pro přenos signálu bude použita plocha automatu bez pomocných konstrukcí, které by simulovaly vodiče. Tento fakt je ale vůči konstrukci irelevantní, protože kluzáky, použité jako základní jednotka informace, se mohou pohybovat pouze diagonálně ve čtyřech směrech. Jako generátor základních logických hodin lze chápat periodu s jakou generuje Gosperův glider gun kluzáky. Pokud kluzák bude chápán jako signál log 1, pak nepřítomnost kluzáku bude chápána jako signál log 0 [12]. Tohoto lze využít pro konstrukci logického hradla NOT. Signál, který má být negován nastavíme ke koliznímu kurzu s trajektorií kluzáků generovaných pomocným Gosperovým dělem. Výsledkem této kolize bude negovaný původní signál, protože pokud byl původní signál log 1, dojde při správně zvolené poloze děla ke kolizi kluzáků, přičemž oba zaniknou, a výsledkem bude signál log 0. Pokud bude původní signál log 0, kluzáky generované pomocným dělem budou pokračovat dál bez kolize (ale ve směru otočeném o 90°). Konstrukce logického hradla AND bude založena na podobném principu jako logické hradlo NOT. Výsledek logického násobení nabývá log 1 pouze v případě, že oba logické vstupy jsou singály log 1. Pokud tedy první logický signál necháme kolidovat s kluzáky generovanými z přesně umístěného Gosperova děla, dostáváme výsledek log členu NOT. Tento výsledek necháme následně kolidovat s druhým vstupním signálem. Pokud byl první vstupní signál log 1, po první kolizi bude výsledek log 0 a druhý vstupní signál projde do výstupu beze změny. Pokud byl první log signál log 0, výsledek po první kolizi bude log 1. Pokud byl druhý logický signál log 1, dojde ke kolizi a výstup bude log 0. Pokud byl log 0, ke kolizi nedojde, ale výstup první kolize bude pokračovat v log 1 ve špatném směru a tudíž na celkovém výstupu bude opět log 0. Pokud budou oba singály v log 0, kluzáky generované pomocným dělem projdou beze změny, ve směru jiném než je ten celkového výstupu. Pro zabránění šíření kluzáků ve směru jiném než je výstupem požadovaný obsahuje obvod eater objekt, který eliminuje všechny kluzáky co nekolidovaly v místě druhé kolize



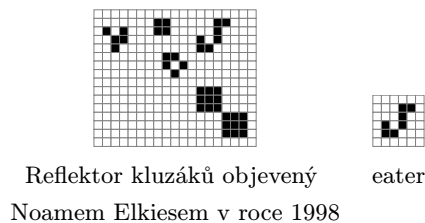
Obrázek 3.3: Logické hradla konstruované v celulárním automatu Game of Life



Obrázek 3.4: Dělič frekvence generování kluzáků

(z důvodu log 0 druhého vstupního signálu). Tohoto je využito pro konstrukci logického hradla OR. Právě v místě, kde je pozicován eater se nachází log signál, který je v log 1 pokud nedošlo k žádné kolizi (oba signály jsou nulové) a signál tedy odpovídá logickému členu NOR. Přidání pomocného Gosperova děla je pak docíleno výsledku, který funguje jako log hradlo OR. Princip logických hradel popsaných výše je zobrazen na obrázku 3.3.

Z teoretického hlediska je konstrukce logického obvodu metodou popsanou výše v pořádku, ale realizace celé konstrukce trpí zásadním nedostatkem, a to sice rozsahem velikosti kolize kluzáků. Kolize dvou kluzáků zasáhne i další připlouvající kluzák [12]. Proto je potřeba zmenšit frekvenci se kterou kluzákové dělo generuje kluzáky. Toto lze vyřešit jakýmsi děličem kmitočtu poskládaným ze dvou dalších pomocných děl. Princip je vyobrazen na obr. 3.4. Dvě děla jsou paralelně vedle sebe ale každé generuje kluzáky v jiném směru. Mezi trajektoriemi kluzáků generovaných děly se pohybuje samostatný kluzák tak, že při kolizi dojde ke změně jeho směru, přičemž generovaný kluzák zanikne a vytvoří blok. Tento blok je pak zničen dalším generovaným kluzákem. Vznikne tak mezera v proudu kluzáků generovaných horním dělem, kterou může proletět kluzák generovaný prostředním dělem. Jako pomocný objekt obvodu může sloužit reflektor, což je objekt (nebo seskupení objektů) měnící směr pohybu kluzáku. Na obrázku 3.5 je uveden příklad takového objektu (resp. jejich seskupení). Pro Turingovskou úplnost potřebuje konstruovaný automat dále nekonečně velkou paměť. K tomu lze použít posuvný paměťový blok, navržený Deanem Hickersonem [4]. Princip spočívá v kolizi dvou kluzáků s blokem, čímž se blok posouvá o pozici dál. Pozice bloku pak určuje nenulové číslo. Další proud gliderů na nulové pozici testuje, zda je registr vynulován či nikoliv. Tyto tři operace spolu s hradlovou logikou zmíněnou výše jsou dostačující ke konstrukci Registrového stroje, který je - jak dokázal Minsky - Turingovsky univerzální [14].



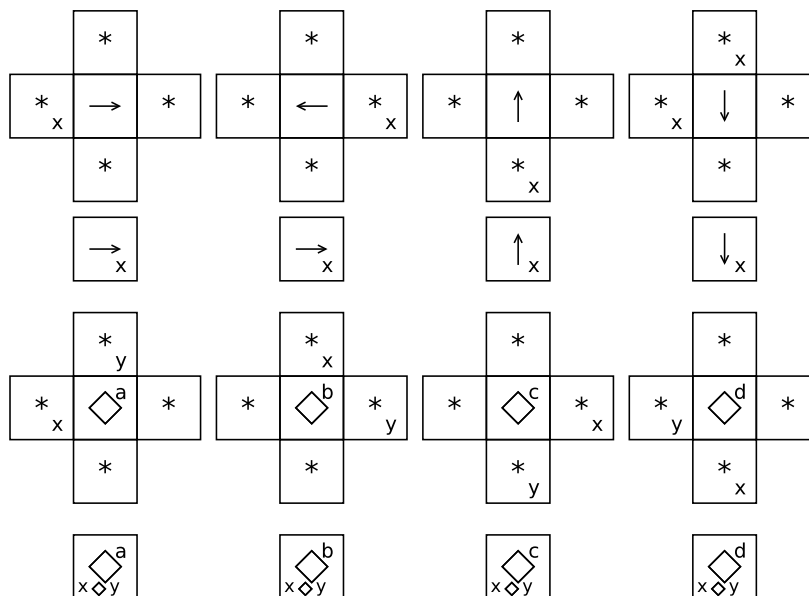
Obrázek 3.5: Reflektor kluzáků a eater objekt

3.4 Univerzální výpočet na 2D neuniformním celulárním automatu

Na neuniformním automatu bude naznačena schopnost univerzálního výpočtu podobným způsobem, jako výše popsané počítání pomocí logických obvodů na automatu Game of Life. Možnost konfigurace přechodových pravidel pro každou buňku zvláště tento úkol značně zjednoduší. Nebude potřeba speciálních objektů jako nositelů základní jednotky informace a signály budou putovat po “drátech”, které budou vymezeny speciální buňkou, nikoliv možným směrem pohybu kluzáků, jako tomu bylo v předchozím případě. Na neuniverzálním automatu je možné také velmi snadno implementovat větvení a křížení signálu. Za poznámku ovšem stojí, že takto konstruovaný automat bude za cenu jednoduchosti sestávat z většího množství pravidel a možných stavů buněk. Konstrukce se bude držet principu univerzálního automatu M. Sippera.²

Pro přenos signálu bude jako základní stavební součást potřeba jakéhosi nosiče informace, tedy drátu, po kterém bude signál putovat. Stejně jako v předchozím případě bude moci signál putovat čtyřmi různými směry. Logický signál bude přenášen pomocí buněk ve stavu 1 a 0. Jediný další potřebný stav bude označení prázdné buňky. Další označení buněk nebude značit jejich stav, ale množinu pravidel přidělených buňce (například množina pro XOR dvou sousedních buněk). Přechodová pravidla použitá pro buňky jsou zobrazena v tabulce na obr. 3.6. Z obrazku je patrné, že automat využívá pouze 5-okolí buňky. Pravidla pro přenos signálu akceptují na každé pravidlo pouze jednu možnou pozici buňky z níž je signál propagován. Aktivní buňky akceptují buňky operandů ve čtyřech směrech s omezením nutného sousedění operandů po diagonále. Za pomoci hradel XOR a NAND lze realizovat všechny ostatní logické funkce. Pomocí buněk vedoucích signál jej lze snadno rozvětvit do více signálových vodičů. Křížení vodičů lze realizovat složením více hradel XOR. Nejdříve dva vstupní signály zpracuje první XOR buňka. Takto zpracovaný signál putuje do další XOR buňky jako první vstup, zatímco jeden z původních dvou signálů jako vstup druhý. Chování celkového výstupu pak odpovídá $((x \otimes y) \otimes x)$, což po úpravě dává y . Výše zmíněné pravidla postačují pro konstrukci automatu Turingovsky univerzálního. Pro emulaci pásky Turingova stroje je ovšem zapotřebí nekonečná paměť, a proto je potřeba automat konstruovat s nekonečnou počáteční konfigurací [18].

²Nicméně jak sám autor přiznává, práce zabývající se konstrukcí automatů podobným způsobem již existují.



Obrázek 3.6: Tabulka pravidel pro neuniformní univerzální celulární automat

Symbol uprostřed označuje množinu pravidel pro danou buňku (spolu s horním indexem - pokud je zahrnut), * značí libovolnou množinu pravidel a libovolný stav (v dané množině pravidel výsledek na buňce nezáleží). Dolní index označuje aktuální stav buňky. Pokud není dolní index přítomen, buňka se nachází v prázdném stavu. Symbol \diamond značí zobecnění pro buňku typu XOR nebo NAND, x a y značí buňky ve stavu 1 nebo 0. Symboly $\leftarrow\rightarrow\uparrow\downarrow$ značí buňky propagující signál [18]

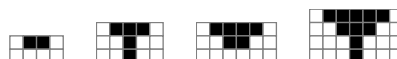
3.5 Další možnosti počítání na celulárních automatech

3.5.1 Počítání na 1D celulárních automatech

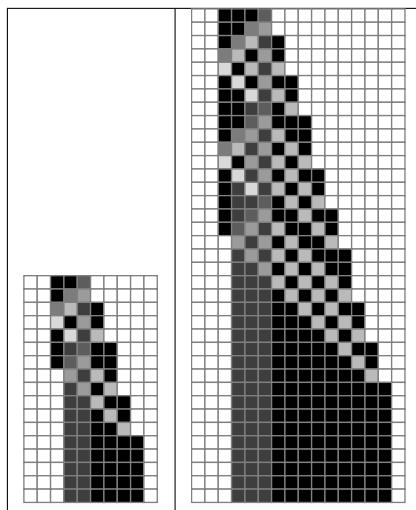
Mimo simulaci Turingova stroje lze na 1D celulárním automatu provádět specifické výpočty způsobem, který popisuje S. Wolfram v [20]. Použité principy pracují s číselným vstupem, které je na plochu automatu zakódováno zvoleným způsobem v unární podobě. Na obr.3.7 je ukázáno použití pravidla 132, které funguje jako detektor lichých čísel. Na obr.3.8 je vykreslen průběh konfigurace pro spočtení mocniny čísla. Na obou příkladech je vidět stoupající doba výpočtu v důsledku velikosti vstupní hodnoty. Zatímco výpočet pro mocninu čísla 2 potřeboval 13 kroků, pro výpočet mocniny čísla 3 je tento počet víc než dvojnásobný (29).

3.5.2 Binární počítání na 2D celulárních automatech

Způsob binárního sčítání a násobení založeného na kolizi uvedli v [15] Petragilio, Tempesti a Henry. Princip binárního sčítání spočívá v kolizi jednotlivých bitů proudu binárních čísel v místě výpočetní buňky. Buňky obou proudu zanikají a vzniká nová buňka obsahující vý-

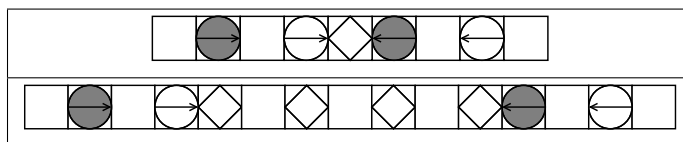


Obrázek 3.7: Pravidlo 132 jako dektor lichých čísel. Prostřední buňka v posledním kroku určuje zda je číslo liché nebo sudé



Obrázek 3.8: 1D automat počítající mocninu čísla

Vpravo je vyobrazen průběh konfigurací počítající mocninu čísla 2, vlevo pak mocninu čísla 3. Hodnota čísla je reprezentována posloupností černých buněk. Automat rozlišuje 8 barevně odstupňovaných stavů.



Obrázek 3.9: binární operace na 2d celulárních automatech

Šipka určuje směr putování proudu buněk. \bigcirc určuje buňku v proudu nesoucí informaci, rozlišenou podle barvy. \diamond značí procesní buňku. Na první řádce je vyobrazena operace sčítání, na druhém násobení.

sledek putující doleva. Bitové řetězce jsou uspořádány tak, že nejméně významné bity obou operandů vstupují do kolize první. Procesní jednotka si uchovává informaci o přenosovém bitu, kterou použije při výpočtu výsledku dalších buněk v kolizi³.

Binární násobení funguje na obdobném principu ovšem na skupině procesních buněk. Tyto buňky si v sobě uchovávají informaci o výsledku, buňky obou proudu nezanikají, ale pokračují dál. Výsledek násobení je pak reprezentován obsahem procesních buněk. Principy jsou vyobrazeny na obr.3.9

3.5.3 Specializované počítání na celulárních automatech

Dosud zmíněné způsoby počítání na celulárních automatech v této kapitole vykazují schopnost počítat s libvolně velkými operandy na základě systematicky navržených pravidel přechodové funkce. Problém systematických řešení spočívá v nesnadnosti jejich objevení. Pokud je maximální počet bitů operandů omezen, lze nalézt řešení pracující s přechodovou funkcí navrženou nesystematicky. Omezením pro takto nalezené řešení mimo šířky operandů může

³ Autoři se v původní práci odkazují na částicový model navržený Steiglitzem bez uvedení podrobnějších detailů. Zjevně, aby uvedený model fungoval, kolize probíhá pouze v okolí procesní buňky a mimo ni se částice proudů jdoucí proti sobě „minou“ beze změny (buňky se prohodí).

být i závislost na zvolených okrajových podmínkách a na rozměrech mřížky CA. Při hledání nesystematického řešení je třeba provést následující kroky:

- Stanovit rozměry a okrajové podmínky CA.
- Omezit maximální bitové šířky operandů.
- Zvolit vhodné zakódování vstupních operandů a výsledku.
- Zvolit optimalizační a prohledávací funkce pro nalezení výsledku.

Příklad zakódování vstupů a výstupů je uveden na obr. 3.10. Buňky označené jako A_x a B_x jsou buňky obsahující vstupní operandy A a B . Buňky označené jako C_x jsou buňky, kde je očekáván výsledek operace po konečném počtu kroků CA. Při hledání řešení je konfigurace CA nastavena na homogení stav - při dvoustavovém CA obvykle 0. Následně jsou buňky určené pro operandy nastaveny do patřičných stavů. Po konečném počtu kroků je výsledek obsažen v buňkách označených jako C_x přičemž na obsahu ostatních buněk nezáleží. Výše popsaný proces se provede pro všechny kombinace vstupních operandů. Pokud je po určitém konečném počtu kroků v buňkách C_x správný výsledek pro všechny kombinace, nalezené řešení je správné.

			C1	
	A1	B1		
			C2	
	A2	B2		
			C3	

Obrázek 3.10: Příklad zakódování vstupů a výstupů do CA pro nalezení nesystematického řešení

V [2] je hledání nesystematicky nalezených řešení ukázáno na příkladech násobení dvou dvoubitových čísel. Za použití vhodně zvoleného zakódování vstupů a výstupů do matice CA autor dokázal, že existuje vysoká pravděpodobnost nalezení řešení pro daný problém. Jako optimalizační a prohledávací funkci použil autor genetický algoritmus, který patří do skupiny evolučních algoritmů. Protože tato práce má podobné zaměření, bude konkrétní volba zakódování vstupů a výstupů a aplikace vybraného evolučního algoritmu popsána podrobněji v následujících kapitolách.

Kapitola 4

Evoluční algoritmy

Herbert Spencer, který měl významné zásluhy při popularizaci Darwinovy teorie, vymyslel pro tyto účely frázi „nejlepší přežije“. Výhodou této fráze je, že každého přesvědčí, že vlastně rozumí Darwinově teorii, a její nevýhodou je, že každého přesvědčí, že vlastně rozumí Darwinově teorii [1].

Základní myšlenka evolučních algoritmů spočívá v použití procesu podobnému evoluci biologických organismů, k postupnému zlepšování evolučně konstruovaného produktu za účelem dosažení předem určeného cíle. Takovýto proces je možné použít ke zlepšení již existujícího objektu, případně doladění polotovaru do potřebného stavu nebo k vytvoření objektu úplně nového. Výsledkem evolučního procesu bývají řešení, která nejsou založena na konvenčních pravidlech, může být velmi nesnadné pochopit jak pracují a často v některých vlastnostech předčí běžná systematicky vytvořená řešení.

Adrian Thompson aplikoval evoluční algoritmus na vývoj elektronického obvodu. Podrobný popis jeho experimentu lze nalézt v [19]. Jeho cíl byl navrhnout obvod s jednoduchým účelem - šit signál 1 KHz od signálu 10 KHz. Požadované výstupy obvodu byla dvě stabilní odlišná napětí pro různé vstupní kmitočty. Thompson průběh algoritmu nesimuloval pouze teoreticky na počítači, ale výsledky vytvářel přímo na programovatelném hradlovém poli. Perfektního řešení dosáhl až po 4 100 generacích vývoje s výstupy 0 voltů pro 10 KHz vstup a 5 voltů pro 1 KHz vstup. Jeho výsledek spotřeboval pouze 100 logických hradel a byl z konceptuálního hlediska principiálně naprosto nepochopitelný. Ovšem při přenosu do počítačem simulované podoby výsledek nefungoval. Jak se ukázalo - navržené řešení nejenže dosahovalo extrémně malých rozměrů (později bylo dokázáno, že ke skutečnému fungování evolucí navrženého obvodu je potřeba pouhých 32 logických hradel), ale používalo i elektrické vlastnosti, které by v simulaci nebyly uvažovány, jako například parazitní kapacita[1].

4.1 Princip evolučních algoritmů

Průběh a princip evolučního algoritmu se dá shrnout do sekvence následujících kroků:

1. Vytvoření počáteční generace jedinců zastupujících varianty aktuálně navržené množiny řešení.
2. Ohodnocení jedinců populace.
3. Pokud není nalezeno následuje cyklus kroků:

- (a) Výběr nejvhodnějších jedinců stávající populace
- (b) Vytvoření nové populace aplikací vhodných operátorů na vybrané jedince
- (c) Ohodnocení jedinců nové populace
- (d) Nahrazení stávající populace novou populací.

Před započítáním evolučního algoritmu je potřeba stanovit základní stavební prvky konstruovaného jedince a jeho možné platné hodnoty. Tento fakt omezí vliv evolučního vývoje na míru, která je únosná pro prostředí v němž je řešení navrhováno. Například pro Thompsonův experiment byla tato hranice stanovena na logické hradlo. Evoluční proces mohl měnit zapojení logických hradel, ale nemohl změnit fakt, že celkový obvod bude sestaven z existujících logických hradel a ne nově vytvořených úspořádání křemíkové struktury. Dalším krokem, bezpodmínečně nutným k provedení před samotným evolučním procesem, je popsání stavebního prostředí a stavební struktury jedince. Tímto je určeno z jakých částí se výsledný produkt bude skládat a jak toho bude docíleno. Na příkladu Thompsonova experimentu je toto omezení stanoveno faktem, že obvod musí mít nějaké vstupy, nějaké výstupy a že propojení jednotlivých hradel lze provést pouze určitým způsobem, který je daný konstrukcí hradlového pole. Takto definovanou strukturu objektu je vhodné popsat způsobem, na který lze aplikovat genetické operace, které jsou popsány níže. Nyní bude definována terminologie používaná v oblasti biologie způsobem, který je aplikován na objekty vytvářené genetickým algoritmem.

populace Populací se rozumí vybraná množina jedinců (řešení), zpravidla s limitovanou velikostí. Populace sestává z aktuálně vzniklých jedinců směřujících ke splnění stanoveného cíle.

chromozom Řetězec popisující stavbu a úspořádání stavebních bloků jedince (na příkladu Thompsonova experimentu je to popis všech hradel programovatelného pole). Chromozom může být koncepčně rozdělen do genů [13].

gen Základní blok popisující stavbu jedince. Určuje, zda bude mít jedinec danou vlastnost či nikoliv. V dalším textu bude gen referovat použití konkrétního základního stavebního bloku (hradlo je použito - gen je obsažen).

alela Konkrétní možná hodnota vlastnosti genu (například modrá barva očí) [13]. Z pohledu popisu stavby jedince řetězcem je to konkrétní hodnota, kterou může prvek řetězce nabývat (např. 0 nebo 1 z pohledu binárního řetězce).

genotyp Souhrn všech genetických informací jedince (všechny možnosti, kterými může být hradlové pole konfigurováno).

generace Konkrétní množina jedinců, kteří tvoří populaci v určitém generačním kroku.

fitness Hodnota určující do jaké míry splňuje jedinec stanovený cíl. Metoda pro stanovení této hodnoty je fitness funkce.

Po tomto kroku už lze zvolit počáteční populaci jedinců. Ta může sestávat z náhodně vytvořených jedinců, tj. jedinců jejichž struktura byla náhodně zvolena s ohledem na omezující podmínky okolí a na výše popsané požadavky, nebo stejných jedinců modifikovaných pouze v určitých rysech. Následně je potřeba populaci ohodnotit - do jaké míry splňují jedinci požadovaný cíl - tj. přiřazení fitness hodnoty každému jedinci z populace. Tento krok je

z obecného pohledu velmi problematický, viz. citát na začátku kapitoly. Z dlouhodobého pohledu není možné stanovit jaký vliv bude mít konkrétní daný gen jedince na splnění požadovaného cíle. A protože je velikost hodnocené populace omezená, nelze prozkoumat celý genotyp. Vzhledem k obrovským prostorovým a časovým nárokům, které by si prohledání celého genotypu vynutilo, by mohlo být nahrazeno totálním prohledáním možného stavového prostoru a i samotný proces evolučního algoritmu by ztrácel význam.

Hodnota fitness často reprezentuje kombinaci několika kritérií. Po ohodnocení jedinců jsou vybráni jedinci ze současné populace jako základ nové generace. Na tyto jedince jsou pak aplikovány genetické operace křížení a mutace za účelem vytvoření nových řešení. Strategii pro výběr vhodných jedinců pro základ nové populace je velké množství a jejich volba může zásadně ovlivnit rychlost, s jakou evoluční algoritmus dosáhne požadovaného cíle. Často volenými strategiemi jsou turnaj, elita a ruleta. Strategie turnaje spočívá v “zápasu” náhodně zvolených dvojic jedinců současné populace, přičemž vyhrává jedinec mající vyšší ohodnocení fitness. Vítězové turnaje se podílí na tvorbě nové generace. Tímto způsobem je šance že se do nové generace nedostanou pouze jedinci s nejvyšším ohodnocením fitness. Opakem turnaje je elita, kdy právě skupina jedinců s nejvyšší fitness hodnotou je vybrána jako základ pro vytvoření nové populace. Kompromisem těchto dvou metod je ruleta, která vybírá jedince do nové populace s pravděpodobností určenou na základě ohodnocení fitness funkcí. Jedinci s nižší fitness funkcí zabírají menší plochu na kole rulety a při roztočení kole je tak menší šance, že se terčík zastaví právě na ploše jim přidělené. Tento výběr by měl imitovat přirozený výběr popsany Darwinem v teorii o původu druhů [7].

Genetické operace aplikované na skupinu vybraných jedinců jsou mutace a křížení. Mutací se rozumí aplikace náhodného vlivu na chromozom jedince. Z praktického hlediska to znamená změnu hodnoty základního prvku genu (například u genetického algoritmu, kde bude chromozom binární řetěz, přehození 0 za 1 a obráceně). To, který prvek bude měněn, je určeno náhodně. Jaký efekt tato změna způsobí, záleží pouze na volbě zakódování struktury jedince do chromozomu. Samozřejmostí je kontrola a eliminace všech zakázaných stavů, do nichž by se mohl jedinec reprezentovaný mutovaným chromozomem dostat (např. pokud hodnota genu může nabývat hodnot od 0 do 4, nelze považovat jedince s hodnotou daného genu 5 za přijatelného). Křížením je chápáno rozdělení chromozomů dvou jedinců do několika částí a jejich následné zkombinování. Zkombinováním částí chromozomů nesmí vzniknout nový genotyp (nelze zkombinovat 2 poloviny chromozomu tak, aby výsledný chromozom kupříkladu neměl ve svém popisu žádné vstupy obvodu ale pouze výstupy). To, kolik a jak velkých částí bude a kolik jedinců jejich kombinací vznikne, záleží na volbě strategie křížení. Nejjednodušší volbou je rozdělení chromozomů jedinců na dvě části a zkombinování první částí od prvního jedince s druhou částí od druhého jedince[8]. Provedením těchto genetických operací dostáváme populaci nových jedinců s potenciálem lepšího ohodnocení fitness.

Popsanou sekvenci lze opakovat až do dosažení požadovaného cíle. Hlavním úskalím použití evolučního algoritmu je jeho závislost na náhodě. To vede hned ke dvěma negativním vlastnostem, kvůli kterým je nasazování evolučních algoritmů do praktických situací omezeno. Evoluční algoritmus může jako výsledek vytvořit velmi originální a efektivní řešení, nicméně není stanovena doba, za kterou toto řešení bude nalezeno, a už vůbec není garantováno, že řešení splňující požadované podmínky vůbec nalezeno bude. Další negativní aspekt vyplývá z výsledku Thompsonova experimentu - nalezené řešení (pokud není pouze simulováno) může být závislé na konkrétním daném prostředí, ve kterém evoluce probíhá, a nemusí být přenositelné do jiného prostředí. S ohledem na nekonvečnost nalezených řešení nemusí být vždy snadné toto řešení upravit tak, aby bylo přenositelné.

Do skupiny evolučních algoritmů patří z nejznámějších například: evoluční programování, evoluční strategie, genetické programování a genetické algoritmy.

Evoluční strategie a také evoluční programování pracují s parametry v reálných číslech. Zakódování genotypu je tedy reprezentováno jako vektor reálných čísel. Mutace řešení je provedena vygenerováním vektoru stejné délky jako vektor reprezentující genotyp s hodnotami náhodně vybranými z gausovského rozložení pravděpodobnosti se středem v 0 a odchylkou 0,5 [8]. Takto vzniklý vektor se přičte k vektoru reprezentujícímu genotyp čímž vznikne nové řešení. Křížení dvou jedinců je opět realizováno operacemi nad vektory, které je reprezentují. Možností je zde mnoho. Diskrétní křížení například vezme určité hodnoty z vektoru jednoho rodiče a doplní je zbylými hodnotami z vektoru druhého rodiče. Průměrové křížení vezme oba vektory rodičů, sečte je a podělí 2, čímž získá vektor, který je průměrem hodnot původních vektorů. Evoluční strategie se ze svého principu nejčastěji nasazují při hledání řešení úloh, které pracují s reálnými čísly.

Genetické programování je optimalizační algoritmus, ve kterém kandidátní řešení reprezentuje hledaný program. Oproti evolučním strategiím a genetickému algoritmu nemá zakódované řešení genetického programování pevnou délku. Zadokování je možné realizovat stromem, lineární reprezentací a nebo grafem [8]. Už ze své podstaty většinou nemají řešení GP stejný genotyp, ale genotyp a fenotyp pro konkrétní řešení splývají. Velikost genotypu je pak omezena například maximální hloubkou stromu (pokud je řešení zakódováno formou stromu). Strom reprezentující řešení GP se může skládat ze dvou základních uzlů, kterými jsou terminály a funkce. Pod pojmem terminály jsou shrnuty veškeré možné konstrukce, které jazyk podporuje, a které neakceptují žádný další argument - pro uzel nelze vytvořit podstrom. Z nejběžněji používaných konstrukcí se jedná například o konstanty, proměnné reprezentující vstup či funkce bez parametru. Funkční uzel pod sebou zastřešuje veškeré jazykové prostředky, které přijímají parametry, a lze u nich vytvořit podstrom. Jsou to například funkce s parametry, skoky, podmínky a cykly. Mutace u GP probíhá náhodným výběrem uzlu ve stromu a jeho nahrazením náhodně vygenerovaným podstromem. Křížení je nejčastěji realizováno jako záměna dvou náhodně vybraných podstromů ze dvou rodičů.

Genetický algoritmus - obdobně jako evoluční strategie - je evoluční algoritmus pro vyhledávání optimálních parametrů (hodnot genů). Stejně tak má i pevnou délku zakódovaného řešení a stejný genotyp pro všechna řešení. Narozdíl od evoluční strategie však nepracuje s reálnými čísly, nýbrž s celými. Mutace u GA probíhá změnou základní části zakódovaného řetězce řešení. Při binárním kódování, kde je základním prvkem řetězce jeden bit, znamená tato změna negaci vybraného bitu. Křížení je obdobné jako u evolučních strategií s tím rozdílem, že způsob zakódování neumožňuje provádět aritmetický průměr rodičů, ale pouze záměnu částí řetězců mezi rodiči. Počet a velikost zaměněných částí jsou závislé na volbě počtu křížících bodů. Nejčastější volbou jsou jednobodové a vícebodové křížení, v nichž jsou vybrány a zaměněny navzájem souvislé části řetězců rodičů. Větší dopad může mít uniformní křížení, kdy jsou dva bity (při binárním zakódování řetězce) na stejných pozicích zaměněny s určitou pravděpodobností.

4.2 Genetický algoritmus při konstrukci celulárního automatu

Konstrukce celulárního automatu tak, aby splňoval stanovený cíl, se stává v důsledku obrovského stavového prostoru mnohdy velmi nesnadnou, pro manuální návrh až téměř nemožnou. Z tohoto důvodu se často nasazuje k vývoji specifických celulárních automatů genetický

algoritmus, který vykoná většinu „hrubé síly“. Z pohledu genetického algoritmu je tedy genotyp kombinace veškerých možných konfigurací okolí a výstupů, kterých může přechodová funkce nabývat. Chromozom pak odpovídá konkrétní kombinaci použitých přechodových pravidel. Gen lze chápat jako přechodové pravidlo pro konkrétní vstup přechodové funkce a alela specifikuje k danému konkrétnímu přechodovému pravidlu konkrétní výstup.

Úskalí evolučního návrhu celulárního automatu spočívá ve vyhodnocování kandidátních řešení. Vyhodnocení celulárního automatu je nutné provést pro všechny požadované vstupní kombinace. Již z definice není výpočet celulárního automatu nijak implicitně ukončen jako je tomu například u Turingova stroje. Výše (sekce 3.2) v popisu jeho simulace byl výpočet ukončen přepsáním symbolu reprezentujícím hlavu pomocným symbolem, čímž se automat dostal do stabilní konfigurace. Na toto ale nelze při evoluci spoléhat - automat může obsahovat vzory, které budou oscilovat do nekonečna. Dalším problémem je stanovení minimálního počtu kroků nutných k docílení požadované výstupní konfigurace. I při nastavení stabilní konfigurace jako ukončovací podmínky výpočtu není zřejmé, kolik kroků bude nutných k docílení takovéto konfigurace. A už vůbec není zaručeno, že minimální počet kroků potřebných k docílení výstupní konfigurace bude neměnný vůči různým vstupním konfiguracím. Zmíněné problémy znesnadňují použití evolučních algoritmů při vývoji celulárních automatů, a proto je potřeba k jejich eliminaci stanovit omezující podmínky, které ovšem můžou mít za důsledek menší efektivitu při hledání řešení.

První možností, jak omezit vývoj CA, je stanovení pevného počtu kroků, po kterých konfigurace automatu musí splňovat všechny podmínky. Výsledné řešení bude tímto pevně omezeno. Avšak i při stanovení pevného počtu kroků lze najít řešení, které splní všechny podmínky po menším počtu kroků, a ve zbývajících krocích zůstane konfigurace neměnná. To může být způsobeno buď tím, že na aktuální konfiguraci CA nelze aplikovat žádné pravidlo z přechodové funkce a buňky tedy zachovávají svůj stav, nebo aplikovaná přechodová funkce obsahuje pravidla, pro které je původní a nový stav buňky shodný. Výhodou tohoto omezení je možnost nalézt řešení, které je po určitém počtu kroků stabilní v čase, a není tedy nutné stanovovat explicitní ukončovací podmínku, protože ukončení výpočtu proběhne v době, kdy se konfigurace automatu nebude měnit.

Další možností je zvolit maximální počet kroků, pro které bude konfigurace automatu kontrolována. Jistou nevýhodou tohoto řešení jsou nutnost kontrolovat splnění požadovaných cílových podmínek po každém kroku automatu a nestálost konfigurace v čase. Výhoda tohoto řešení se nalézá v poskytnutí rozmanité škály řešení s různým počtem kroků a v menším počtu populací evoluovaných k nalezení řešení. U každého takového řešení je nutná dodatečná informace o přesném počtu kroků potřebných ke splnění stanovených podmínek. Tato metoda je použita v této práci.

4.2.1 Genetický algoritmus při hledání nesystematického řešení celulárních automatů

Úkolem genetického algoritmu při hledání nesystematického řešení je nalezení takové přechodové funkce, pro kterou budou za určitý konečný počet kroků konfigurace automatu splňovat všechny výstupní podmínky. Jak již bylo řečeno, obecným cílem evolučních algoritmů je optimalizace řešení. Vhodnou volbou parametrů procesu evoluce, se lze přiblížit optimálnímu řešení. Naopak nevhodnou volbou parametrů je možné zapříčinit, že GA nalezne řešení, které optimální nebude. Cílem této práce však není hledání optimalizovaných řešení pro specifické úlohy, ale demonstrace toho, že řešení existuje. Proto volba nastavení parametrů GA, která nemusí být vhodná pro úkoly hledání maxima funkce, může být na-

opak velmi vhodná pro potřeby této práce - a to najít co nejvíce řešení v co nejkratším čase.

Prvním parametrem značně ovlivňujícím průběh GA je volba vhodné metody pro selekci nové populace. Jak již bylo zmíněno, mezi nejběžnější postupy selekce jedinců, kteří se budou podílet na tvorbě nové populace, patří ruleta, elita, turnaj. Zatímco ruleta a turnaj vnášejí do nové populace značnou možnost rozmanitosti řešení, systém elity se jeví jako nejkonzervativnější z těchto přístupů a proto velmi vhodný pro výběr nové populace.

Dalším parametrem, který se významně podílí na průběhu GA je pravděpodobnost mutace. Při hledání optima funkce se hodnota této pravděpodobnosti pohybuje pod 10 % [8]. Při požadavku nalézt řešení co nejrychleji bez ohledu na jeho kvalitu je vhodné tuto pravděpodobnost nastavit na mnohem vyšší hodnotu.

Správnou volbou velikosti populace lze do značné míry ovlivnit efektivitu selekční metody. Dostatečně velká populace jedinců umožňuje při použití ruletové nebo turnajové selekce poskytnout GA velkou diverzitu jedinců a tím vyšší pravděpodobnost nalezení optimálního řešení. Stejně tak u elitismu umožňuje poskytnout dostatečný počet jedinců pro udržení neklesající kvality nejlépe hodnoceného řešení, případně minimalizovat výkyvy této hodnoty.

Počítání fitness pro vyhodnocování řešení CA

Správná volba výpočtu fitness je základním kamenem pro fungování GA. V sekci 4.2 byly zmíněny dvě možnosti vyhodnocování správnosti kandidátních řešení nalezených GA. Z důvodu odlišnosti těchto dvou možností je i způsob, kterým se bude počítat fitness, pro každou z nich jiný. Při vyhodnocování řešení po každém kroku, je potřeba ohodnotit konfigurace CA pro všechny vstupní a výstupní podmínky hledaného automatu. Po kterémkoliv kroku se může dostat kandidátní řešení do konfigurací, které vyhovuje stanoveným podmínkám, a v takovém případě je vráceno jako výsledek. Dalším případem je stagnace řešení, při níž konfigurace automatů zůstanou stabilní v důsledku nemožnosti aplikovat na ně pravidla přechodové funkce nebo v důsledku stejného současného a nového stavu buňky v těchto pravidlech. Pokud dojde k tomu, že všechna řešení v populaci stagnují, nemá smysl dále tuto populaci udržovat při životě a je potřeba přikročit k předčasnému vytvoření nové populace.

Nejintuitivnější hodnotou, která se dá použít k výpočtu fitness funkce, je počet buněk konfigurace CA splňující stanovenou podmínku. Tato hodnota vychází vždy z aktuální konfigurace, tudíž není potřeba uchovávat její historii, a je celočíselná. Usnadňující může být neexistence potřeby porovnávat všechny buňky CA, ale jenom buňky, které mají obsahovat výsledek. Dále se nabízí možnost sběru dalších hodnot, jako jsou počet minimální počet správných stavů, maximální počet správných stavů, průměrný počet správných stavů, kolikrát bylo dosaženo maximálního počtu správných stavů, kolikrát bylo dosaženo nulového počtu správných stavů, počet kroků před stagnací řešení a tak dále. Tyto hodnoty pak mohou sloužit k sestavení výsledné hodnoty fitness.

Celkový pohled na zvolené vyhodnocování a průběh GA

Ilustrace vyhodnocování a průběhu GA jsou vyobrazeny na obr. 4.1. Vygenerovaná populace obsahuje zvolený počet řešení, přičemž pro všechna řešení jsou stanoveny stejné podmínky vyhodnocování správnosti. Pro každé řešení je při generování populace náhodně vygenerována přechodová funkce. Počet pravidel je zvolen jako parametr. Každé řešení obsahuje

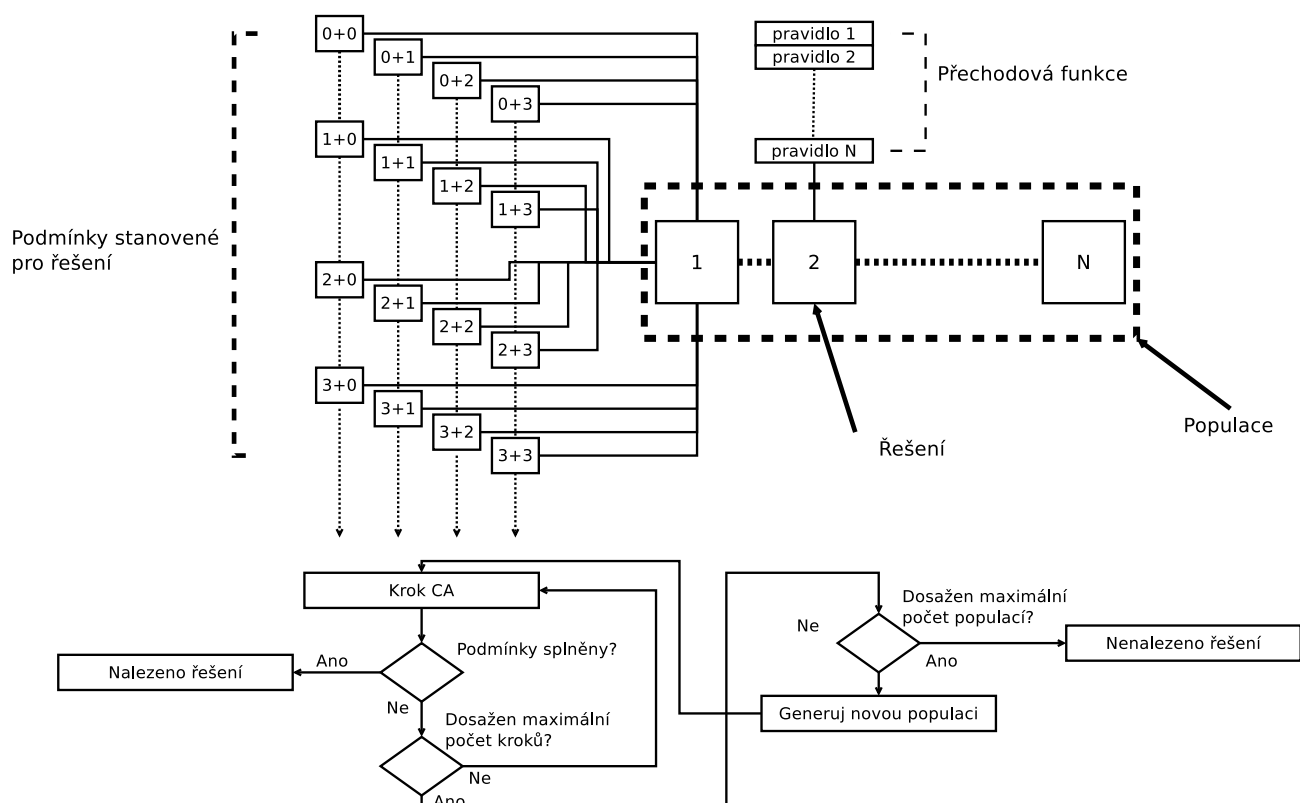
stejný počet matic celulárního automatu jako je počet podmínek. To umožňuje jednodušší vyhodnocování podmínek po každém kroku, a zároveň umožňuje paralelizovat aplikaci kroku CA na jedno řešení. Přejímová funkce je pro dané řešení u všech podmínek stejná. Před započítím průběhu GA jsou všechny CA v podmínkách pro všechna řešení nastaveny na počáteční konfigurace odpovídající daným podmínkám. V průběhu GA je vždy proveden jeden krok pro všechny kandidátní řešení v populaci. Což znamená jeden krok pro všechny CA v podmínkách.

Při provádění kroku je pro každé řešení vyhodnoceno, zda se konfigurace CA změnila nebo ne. Pokud nedojde ke změně konfigurace u ani jedné podmínky, je jedinec označen jako stagnující. Na stagnující jedince není aplikován krok CA a je vynechán z ohodnocení. Což znamená, že poslední hodnoty ohodnocení takového jedince jsou z posledního provedeného ohodnocení, než byl jedinec označen jako stagnující. Toto řešení umožňuje ponechat podmínkám maximální ohodnocení pokud jsou všechny stavy kontrolovaných buněk shodné s požadovanými, ale konfigurace CA už se dál měnit nemůže. Následuje ohodnocení všech podmínkových CA. Jednotlivá ohodnocení podmínek jsou předána odpovídajícímu jedinci. Pro každou podmínku je dán maximální počet buněk, které budou kontrolovány. Odpovídající jedinec pak dostává informaci, kolik podmínek bylo splněno. Při splnění všech podmínek je kandidátní řešení vráceno jako nalezené řešení. Pokud není nalezeno ani jedno správné řešení, pokračuje GA provedením dalšího kroku nad všemi nestagnujícími řešeními následovném ohodnocením. Pokud se všechna řešení dostanou do stagnujícího stavu, přejde se k vytvoření nové populace. Stejná situace nastane, pokud je překročen maximální stanovený počet kroků CA. Při vytváření nové populace jsou řešení stávající populace ohodnocena fitness funkcí, která evaluuje řešení podle hodnot sesbíraných v průběhu sekvence kroků. Po ohodnocení je na řešení aplikována selekční funkce a genetické operátory podle zvolené strategie. Pokud je mutace aplikována na operátor podmínky u podmínkového pravidla, je nový operátor náhodně vybrán z množiny operátorů vybraných pro GA. Po tomto kroku fitness funkce vyprodukuje seznam řešení, která jsou novou populací. Po nastavení počátečních konfigurací CA pro všechny podmínky ve všech řešeních, odstranění stagnujících příznaků a vyresetování hodnot získaných při vyhodnocování, je nová populace připravena k prvnímu kroku CA.

Konkretní parametry genetického algoritmu

V následujícím textu jsou popsány parametry zvolené pro GA. Vzhledem k tomu, že cíle této práce jsou podobné jako v případě [2], jsou některé použité parametry totožné a nebo podobné. Počet podmínkových pravidel byl zachován (15). Stejně tak i množina operátorů podmínkových pravidel. Konkrétně se jedná o tyto operátory: \leq , \geq , \neq , $=$, $*$. Pro úplnost jsou uvedeny pravdivostní tabulky těchto operátorů v tabulce 4.1. Velikost populace je nastavena na 20 kandidátních řešení. Větší počet by mohl urychlit hledání řešení ovšem za cenu větší časové náročnosti k vyhodnocení populace. Maximální počet kroků pro řešení je nastaven na 15 - pro větší počet kroků by automat byl velmi složitý a malá pravděpodobnost nalezení řešení používající takový počet. Jako okolí buňky CA bylo zvoleno Moorovo okolí s poloměrem 1 (tak jako [2]). Okrajové podmínky CA byly zvoleny napevno určené buňky inicializované do stavu 0.

Jako strategie pro selekci řešení do nové populace byla zvolena metoda elitismu. Z důsledku požadavku rychlého hledání řešení, byla pravděpodobnost mutace nastavena na 100 %. Přičemž na všechna řešení vstupující do nové populace je aplikována alespoň 3x (tato hodnota ukázala jako optimální pro zachování fitness hodnoty populace). Operátor



Obrázek 4.1: Celkový pohled na vyhodnocování a průběh GA

Buňka	0	0	1	1		0	0	1	1		0	0	1	1		0	0	1	1
Konstatna	0	1	0	1		0	1	0	1		0	1	0	1		0	1	0	1
Výsledek ≤	1	1	0	1	≥	1	0	1	1	=	1	0	0	1	≠	0	1	1	0
Buňka	0	0	1	1															
Konstatna	0	1	0	1															
Výsledek *	1	1	1	1															

Tabulka 4.1: Pravdivostní tabulky zvolených operátorů

křížení nebyl pro generaci nové populace použit vůbec. Skladba nové populace byla řešena proporčním výběrem. Nejlepší řešení ze současné populace se podílelo na tvorbě $\frac{1}{4}$ nové populace. Z druhého nejlepšího řešení se vytvořila $\frac{1}{5}$ nové populace. Následujícím způsobem se pokračovalo až do doby, kdy počet jedinců nové populace nedosáhl hodnoty 20 (zvolená velikost populace). Každé nové řešení bylo vytvořeno z původního tak, že se na něj aplikoval 3x operátor mutace. Pro seřazení kandidátních řešení od nejlepšího po nejhorší byl použit pouze počet správně splněných podmínek, protože, jak se ukázalo, všechny ostatní parametry (např. kolikrát byla splněna podmínka úplně, kolikrát nebyla splněna podmínka vůbec - všechny kontrolované buňky byly odlišeny, průměrný počet správných buněk v podmínce) sbírané při vyhodnocování podmínek neměly na seřazení, při jejich použití jako dalších klíčů k řazení, vliv.

Kapitola 5

Návrh výpočtů v CA pomocí hledání nesystematického řešení

V této kapitole je popsána realizace zvolených výpočtů pomocí CA. Konkrétní řešení jsou hledána pro sčítání dvou dvoubitových operandů, násobení dvou dvoubitových operandů a hledání druhé mocniny čísla N . V případě sčítání a násobení automat akceptuje maximálně 2 bitové vstupy, výsledek bude tedy maximálně 3 bitový. Tento fakt určuje, že automat musí splnit 16 podmínek pro to, aby sčítání proběhlo správně při jakékoliv kombinaci vstupů. Tento počet není příliš velký a je tedy velmi pravděpodobné, že se v jistých případech podaří nalézt přechové funkce, které budou splňovat požadované chování automatu. V případě násobení je tato situace mírně komplikovanější, jelikož výsledek násobení je 4 bitový. Jak ale dokazuje [2], je násobení na 2D binárním CA možné. V této práci je prozkoumáno, jakých výsledků je možné dosáhnout při zvolení odlišné metody vyhodnocování GA a jiné strategie pro selekci nové populace. Výpočet mocniny komplikuje fakt, že nalézt řešení na binárním CA by bylo značně problematické. V kapitole 3.5.1 je uveden příklad možného výpočtu pro 1D celulární automat, kde je vstupní a výstupní hodnota zakódována unárně jako posloupnost buněk stejného stavu.

5.1 Sčítání v binárním 2D CA

Úkolem pro GA je v tomto případě nalézt takovou přechodovou funkci, aby pro zakódované vstupní operandy po určitém konečném počtu kroků CA obsahoval výstup v buňkách, které pro něj byly určeny. Buňky automatu, na kterém byl prováděn výpočet, mohou obsahovat pouze 2 stavy. Tato skutečnost vybízí ke zvolení binárního kódování pro vstupy a výstup. Automat tedy nedovoluje buňkám obsahovat žádné pomocné stavy, které by se nedaly interpretovat jako výstupní nebo vstupní hodnota. Část vstupních a výstupních konfigurací byla převzata z [2] a část zvolena nově (obr. 5.1).

Zobrazené konfigurace obsahují buňky pro první a druhý vstup do výpočtu označené jako A a B. Číslo u písmena označuje pozici bitu, přičemž 1 je nejméně významný bit. Písmeno C značí buňky určené pro výsledek.

Pro zjištění zda bude konfigurace vhodná k nalezení takové přechové funkce, která po určitém konečném počtu kroků zaručí, že pro libovolný vstup omezený maximálním počtem bitů, bude na výstupu správný výsledek, bylo použito 100 nezávislých běhů GA s limitem 10 000 generací. Tabulka 5.1 zobrazuje počet běhů, ve kterém byl GA úspěšný, dále pak průměrný počet kroků potřebných k dosažení stanovených podmínek, minimální a maxi-

0	0	0	0	0	0	0	0
0							0
0			A1				0
0		A2	C1	B1			0
0		C2	B2				0
0	C3						0
0	0	0	0	0	0	0	0
varianta A							
0	0	0	0	0	0	0	0
0							0
0		A2		A1			0
0			B2		B1		0
0		C3	C2	C1			0
0							0
0	0	0	0	0	0	0	0
varianta B							
0	0	0	0	0	0	0	0
0							0
0		C1	B1	C3	B2		0
0		A1	C2	A2			0
0							0
0							0
0	0	0	0	0	0	0	0
varianta C							
0	0	0	0	0	0	0	0
0							0
0		A1	C1	B1			0
0							0
0							0
0							0
0	0	0	0	0	0	0	0
varianta D							
0	0	0	0	0	0	0	0
0							0
0		A1	C1	B1			0
0							0
0		A2	C2	B2			0
0			C3				0
0	0	0	0	0	0	0	0
varianta E							
0	0	0	0	0	0	0	0
0					C3		0
0		A2	B2				0
0					C2		0
0		A1	B1				0
0					C1		0
0	0	0	0	0	0	0	0

Obrázek 5.1: Volba polohy buněk pro vstupy a výstup CA - sčítání. $C_3C_2C_1 = A_2A_1 + B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.

mální počet kroků k dosažení podmínek, průměrný počet populací nutných k nalezení řešení, minimální a maximální počet generací k nalezení řešení, průměrný počet pravidel použitých v přechodové funkci a minimální a maximální počet pravidel použitý v přechodové funkci.

Varianta	A	B	C	D	E
Počet úspěšných běhů ze 100	85	16	17	97	81
Průměrný počet kroků CA na řešení	6,30	7,68	8,62	6,49	6,80
Minimální počet kroků CA na řešení	3	4	4	3	3
Maximální počet kroků CA na řešení	12	12	14	14	13
Průměrný počet generací k nalezení řešení	1 501,80	5 343,13	5 841,81	892,75	2 166,83
Minimální počet generací k nalazení řešení	42	771	2 965	20	161
Maximální počet generací k nalezení řešení	6 71	9 30	9 993	9 62	8 76
Průměrný počet použitých podm. pravidel	12,67	13,62	13,70	12,69	13,01
Minimální počet použitých podm. pravidel	8	12	10	8	9
Maximální počet použitých podm. pravidel	15	15	15	15	15

Tabulka 5.1: Výsledky 100 běhů pro nalezení přechodové funkce pro sčítání dvoubitových operandů

Protože 3 z 5 variant pro sčítání 2 bitových operandů poskytly velmi úspěšné výsledky (více než 20 řešení), nabízí se otázka, jaké výsledky tyto varianty dosáhnou v případě 3 bitových operandů. Pro 2 bitové operandy musel automat splňovat 16 podmínek správného výsledku po konečném počtu kroků. Po přidání jednoho bitu se zvedne i nutný minimální počet bitů výsledku a počet podmínek se zčtyřnásobí. Na obrázku 5.2 jsou zachyceny úpravené vstupní a výstupní konfigurace. Varianty jsou pojmenovány po předchozích obdobných

variantách pracujících s 2 bitovými operandy.

<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>A1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>A2</td><td>C1</td><td>B1</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A3</td><td>C2</td><td>B2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>C3</td><td>B3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>C4</td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>								0	0	0	0	0	0	0	0	0							0	0				A1			0	0			A2	C1	B1		0	0		A3	C2	B2			0	0		C3	B3				0	0	C4						0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>C1</td><td>B1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>C2</td><td>B2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A3</td><td>C3</td><td>B3</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C4</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>								0	0	0	0	0	0	0	0	0							0	0		A1	C1	B1			0	0							0	0		A2	C2	B2			0	0							0	0		A3	C3	B3			0	0			C4				0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td>C1</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>B1</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>B2</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C3</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A3</td><td>B3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C4</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>								0	0	0	0	0	0	0	0	0					C1		0	0		A1	B1				0	0				C2			0	0		A2	B2				0	0				C3			0	0		A3	B3				0	0				C4			0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
0							0																																																																																																																																																																																																																																
0				A1			0																																																																																																																																																																																																																																
0			A2	C1	B1		0																																																																																																																																																																																																																																
0		A3	C2	B2			0																																																																																																																																																																																																																																
0		C3	B3				0																																																																																																																																																																																																																																
0	C4						0																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
0							0																																																																																																																																																																																																																																
0		A1	C1	B1			0																																																																																																																																																																																																																																
0							0																																																																																																																																																																																																																																
0		A2	C2	B2			0																																																																																																																																																																																																																																
0							0																																																																																																																																																																																																																																
0		A3	C3	B3			0																																																																																																																																																																																																																																
0			C4				0																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
0					C1		0																																																																																																																																																																																																																																
0		A1	B1				0																																																																																																																																																																																																																																
0				C2			0																																																																																																																																																																																																																																
0		A2	B2				0																																																																																																																																																																																																																																
0				C3			0																																																																																																																																																																																																																																
0		A3	B3				0																																																																																																																																																																																																																																
0				C4			0																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																
varianta A								varianta D								varianta E																																																																																																																																																																																																																							

Obrázek 5.2: Volba polohy buněk pro vstupy a výstup CA - sčítání 3 bitových operandů. $C_4C_3C_2C_1 = A_3A_2A_1 + B_3B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.

Pro sadu testů bylo použito opět 100 nezávislých běhů se stejnými parametry GA jako v případě sčítání 2 bitových operandů. Celkové výsledky těchto běhů jsou přehledně zobrazeny v tabulce 5.2.

Varianta	A	D	E
Počet úspěšných běhů	15	67	12
Průměrný počet kroků CA na řešení	9,06	8,34	9,75
Minimální počet kroků CA na řešení	6	5	7
Maximální počet kroků CA na řešení	14	14	13
Průměrný počet generací k nalezení řešení	3 328,00	1 755,71	3 118,82
Minimální počet generací k nalazení řešení	62	89	488
Maximální počet generací k nalezení řešení	5 71	9 89	7 20
Průměrný počet použitých podm. pravidel	13,2	12,95	13,33
Minimální počet použitých podm. pravidel	12	10	10
Maximální počet použitých podm. pravidel	15	15	15

Tabulka 5.2: Výsledky 100 běhů pro nalezení přechodové funkce pro sčítání tříbitových operandů

5.2 Násobení v binárním 2D CA

V [2] bylo dokázáno, že pro 2D binární CA pracující s 9-okolí lze s poměrně vysokou pravděpodobností (až 69 %) nalézt takovou přechodovou funkci, aby po určitém konečném počtu kroků výsledná konfigurace automatu splňovala podmínky dané znásobením dvou vstupních operandů. V uvedené práci bylo použito vyhodnocení řešení s předem stanoveným počtem kroků. Dále bylo požadováno, aby automat po dosažení tohoto počtu kroků setrval ve stabilní konfiguraci. Cílem tohoto výzkumu je zjistit, jaké výsledky budou získány při vyhodnocování CA po každém kroku bez omezení požadavkem na stabilní konfiguraci po ukončení výpočtu. I když jsou konfigurace vstupů a výstupu, na kterých byly experimenty prováděny, podobné s konfiguracemi pro sčítání, díky dalšímu bitu výsledku navíc došlo

k jejich mírné úpravě a proto jsou pro úplnost uvedeny na obr. 5.3. Vyhodnocení 100 nezávislých běhů hledání řešení je uvedeno v tabulce 5.3.

<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>A1</td><td>C1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>C2</td><td>B1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>C3</td><td>B2</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>C4</td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>varianta A</p>	0	0	0	0	0	0	0	0	0							0	0			A1	C1			0	0		A2	C2	B1			0	0		C3	B2				0	0	C4						0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td></td><td>A1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>B2</td><td></td><td>B1</td><td></td><td>0</td></tr><tr><td>0</td><td>C4</td><td>C3</td><td>C2</td><td>C1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>varianta B</p>	0	0	0	0	0	0	0	0	0							0	0		A2		A1			0	0			B2		B1		0	0	C4	C3	C2	C1			0	0							0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>C1</td><td>B1</td><td>C3</td><td>B2</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>C2</td><td>A2</td><td>C4</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>varianta C</p>	0	0	0	0	0	0	0	0	0							0	0		C1	B1	C3	B2		0	0		A1	C2	A2	C4		0	0							0	0							0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0																																																																																																																																																																			
0							0																																																																																																																																																																			
0			A1	C1			0																																																																																																																																																																			
0		A2	C2	B1			0																																																																																																																																																																			
0		C3	B2				0																																																																																																																																																																			
0	C4						0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0							0																																																																																																																																																																			
0		A2		A1			0																																																																																																																																																																			
0			B2		B1		0																																																																																																																																																																			
0	C4	C3	C2	C1			0																																																																																																																																																																			
0							0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0							0																																																																																																																																																																			
0		C1	B1	C3	B2		0																																																																																																																																																																			
0		A1	C2	A2	C4		0																																																																																																																																																																			
0							0																																																																																																																																																																			
0							0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>C1</td><td>B1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C2</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>C3</td><td>B2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C4</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>varianta D</p>	0	0	0	0	0	0	0	0	0							0	0		A1	C1	B1			0	0			C2				0	0		A2	C3	B2			0	0			C4				0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>B2</td><td>C4</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C3</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>B1</td><td>C2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>varianta E</p>	0	0	0	0	0	0	0	0	0							0	0		A2	B2	C4			0	0				C3			0	0		A1	B1	C2			0	0				C1			0	0	0	0	0	0	0	0	0																																																									
0	0	0	0	0	0	0	0																																																																																																																																																																			
0							0																																																																																																																																																																			
0		A1	C1	B1			0																																																																																																																																																																			
0			C2				0																																																																																																																																																																			
0		A2	C3	B2			0																																																																																																																																																																			
0			C4				0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0							0																																																																																																																																																																			
0		A2	B2	C4			0																																																																																																																																																																			
0				C3			0																																																																																																																																																																			
0		A1	B1	C2			0																																																																																																																																																																			
0				C1			0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			

Obrázek 5.3: Volba polohy buněk pro vstupy a výstup CA - násobení. $C_3C_2C_1 = A_2A_1 * B_2B_1$. Buňky s hodnotou 0 značí okrajové buňky, které nemění svou hodnotu v průběhu kroků CA.

Varianta	A	B	C	D	E
Počet úspěšných běhů	9	13	38	92	44
Průměrný počet kroků CA na řešení	7,55	6,76	5,34	4,71	6,38
Maximální počet kroků CA na řešení	13	12	12	13	11
Minimální počet kroků CA na řešení	4	4	3	2	3
Průměrný počet generací k nalezení řešení	4 350,66	4 357,84	4 298,07	1 646,66	4 131,13
Minimální počet generací k nalezení řešení	1 299	170	90	60	249
Maximální počet generací k nalezení řešení	8 131	9 621	9 874	9 266	9 913
Průměrný počet použitých podm. pravidel	12,11	12,46	11,65	11,85	12,25
Minimální počet použitých podm. pravidel	10	10	8	8	8
Maximální počet použitých podm. pravidel	14	14	15	15	15

Tabulka 5.3: Výsledky 100 běhů pro nalezení přechodové funkce pro násobení dvoubitových operandů

Stejně jako u sčítání se ukázalo, že pro některé varianty je vysoká pravděpodobnost nalezení výsledku. Po upravení variant, tak aby na nich bylo možné realizovat násobení 3 bitových operandů, bylo spuštěno 100 nezávislých běhů. Jak se ukázalo, přidání dalších 2 bitů výsledku má za následek selhání evolučního algoritmu, jelikož ani pro jednu variantu se nepodařilo nalézt řešení. Modifikované varianty jsou uvedeny na obr. 5.4 spolu s uvedenými počty podmínek, které se povedlo v průběhu testů splnit.

Počet splněných podmínek:	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>C1</td><td>B1</td><td>C3</td><td>B2</td><td>C5</td><td>B3</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>C2</td><td>A2</td><td>C4</td><td>A3</td><td>C6</td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>										0	0	0	0	0	0	0	0	0	0	0									0	0		C1	B1	C3	B2	C5	B3		0	0		A1	C2	A2	C4	A3	C6		0	0									0	0									0	0									0	0	0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>C1</td><td>B1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C2</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>C3</td><td>B2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C4</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A3</td><td>C4</td><td>B3</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td>C5</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>								0	0	0	0	0	0	0	0	0							0	0		A1	C1	B1			0	0			C2				0	0		A2	C3	B2			0	0			C4				0	0		A3	C4	B3			0	0			C5				0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A3</td><td>B3</td><td>C6</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C5</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A2</td><td>B2</td><td>C4</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C3</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td>A1</td><td>B1</td><td>C2</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td></td><td></td><td></td><td>C1</td><td></td><td></td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>								0	0	0	0	0	0	0	0	0							0	0		A3	B3	C6			0	0				C5			0	0		A2	B2	C4			0	0				C3			0	0		A1	B1	C2			0	0				C1			0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																
	0									0																																																																																																																																																																																																																																																
	0		C1	B1	C3	B2	C5	B3		0																																																																																																																																																																																																																																																
	0		A1	C2	A2	C4	A3	C6		0																																																																																																																																																																																																																																																
	0									0																																																																																																																																																																																																																																																
	0									0																																																																																																																																																																																																																																																
	0									0																																																																																																																																																																																																																																																
	0	0	0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																
	0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																		
0							0																																																																																																																																																																																																																																																			
0		A1	C1	B1			0																																																																																																																																																																																																																																																			
0			C2				0																																																																																																																																																																																																																																																			
0		A2	C3	B2			0																																																																																																																																																																																																																																																			
0			C4				0																																																																																																																																																																																																																																																			
0		A3	C4	B3			0																																																																																																																																																																																																																																																			
0			C5				0																																																																																																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																			
0							0																																																																																																																																																																																																																																																			
0		A3	B3	C6			0																																																																																																																																																																																																																																																			
0				C5			0																																																																																																																																																																																																																																																			
0		A2	B2	C4			0																																																																																																																																																																																																																																																			
0				C3			0																																																																																																																																																																																																																																																			
0		A1	B1	C2			0																																																																																																																																																																																																																																																			
0				C1			0																																																																																																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																			
	Varianta C										Varianta D								Varianta E																																																																																																																																																																																																																																							
Maximální	29/64										48/64								47/64																																																																																																																																																																																																																																							
Průměrný	26,701/64										44,089/64								40,857/64																																																																																																																																																																																																																																							
Minimální	24/64										34/64								32/64																																																																																																																																																																																																																																							

Obrázek 5.4: Upravené varianty pro násobení 3 bitových operandů

5.3 N^2 ve vícestavovém 1D CA

Na obr. 3.8 je ukázáno jak lze spočítat mocninu za pomoci 9 stavového celulárního automatu. Jelikož je výsledek zakódován unárně, situace řešení podmínek je značně odlišná od sčítání nebo násobení popsaného výše. Zatímco u násobení a sčítání byl přesně pro všechny podmínky definován počet bitů výsledku, u mocniny toto neplatí. Z principu použití unárního kódování je stav buňky jako informativní hodnota nedostatečný a místo toho poslouží za výsledek to, kolik buněk po sobě nabývá touto hodnotou. Za poslední buňkou výsledku přirozeně musí následovat buňka s jiným stavem, než je stav nesoucí informaci. Naopak před první buňkou nesoucí informaci nemusí být buňka v jiném stavu, protože počáteční pozici sledu buněk výsledku lze na pevně určit.

Další odlišností oproti sčítání a násobení na 2D CA je počet kroků, které je potřeba vyhradit pro nalezení výsledku. Jak je zmíněno v popisu obr. 3.8, počet kroků se navýšením vstupní hodnoty o 1 více než zdvojnásobí. Za příčinou stojí degradace paralelního zpracování buněk automatu na sekvenční zpracování, kdy v každém kroku automatu mění svůj stav buňka, která je v okolí s jinou vhodnou buňkou a ostatní buňky pouze stagnují. Počet kroků dále navyšuje zvolené kódování, protože počet buněk výsledku roste exponenciálně oproti sčítání a násobení, kde počet buněk výsledku rostl lineárně s počtem buněk potřebných na operand.

Pro potřeby hledání mocniny byl limitní počet kroků upraven na 150 a maximální počet pravidel na 12. Počet stavů, kterých může buňka nabývat byl stanoven na 8.

Úspěšné výsledky genetický algoritmus našel pouze pro mocninu počítající maximálně s hodnotou 3. Tabulka 5.4 zobrazuje souhrné výsledky pro 100 nezávislých běhů experimentu.

Počet úspěšných běhů	41
Průměrný počet kroků CA na řešení	58,26
Maximální počet kroků CA na řešení	146
Minimální počet kroků CA na řešení	5
Průměrný počet generací k nalezení řešení	4 144,29
Minimální počet generací k nalazení řešení	210
Maximální počet generací k nalezení řešení	9 216
Průměrný počet použitých podm. pravidel	7,92
Minimální počet použitých podm. pravidel	5
Maximální počet použitých podm. pravidel	11

Tabulka 5.4: Výsledky 100 běhů pro nalezení přechodové funkce realizující druhou mocninu

Kapitola 6

Experimentální výsledky

V následujícím textu jsou do podrobnosti rozebrána výsledná řešení nalezená genetickým algoritmem. Zejména je porovnávána náročnost nalezených řešení na spotřebované zdroje, jako je počet generací nutných k nalezení, počet používaných podmínkových pravidel a počet kroků potřebných ke splnění všech stanovených podmínek.

6.1 Sčítání 2 bitových operandů

Z tabulky 5.1 je patrné, že většina zvolených vstupních konfigurací je vhodnou volbou pro hledání přechodové funkce, která bude realizovat sčítání. Varianty A, D a E dávají v průměru více než 87 % šanci, že GA nalezne řešení, a to za průměrně asi 1521 generací. Varianty C a D se jeví jako méně vhodné - v průměru dávají pouze 16.5 % podíl úspěšných řešení. Na obrázku 6.1 jsou zobrazeny grafy znázorňující počet generací nutných k nalezení řešení vyjádřených v procentech. Po propojení vykreslených bodů křivkami je patrné, že v případě úspěšných variant křivka vykazuje exponenciální růst, kdežto u méně úspěšných variant má spíše lineární průběh. Nejvíce je tento vztah patrný u varianty D, kde asi 80% z 97 řešení bylo nalezeno ještě před dosažením průměrného počtu generací. Lze pozorovat, že počet generací k nalezení řešení koreluje s počátečním vygenerováním přechodové funkce. Ze skoro lineárních charakteristik variant B a C, lze usoudit, že počet generací, který bude potřebný k nalezení řešení, je do značné míry ovlivněn tím, jak náhodně je přechodová funkce zvolena v první generaci. Zatímco u ostatních úspěšnějších variant je tato míra ovlivnění menší, čím je varianta vhodnější. Tuto domněnku dále utvrzuje to, že u všech variant je více než 60 % řešení nalezeno ještě před průměrným počtem generací. Na obr. 6.2 jsou zachyceny histogramy počtu kroků pro jednotlivé varianty. Jak se ukazuje, minimální počet kroků potřebných ke splnění všech podmínek je v intervalu 3-4 kroků, přičemž počet nalezených řešení pro tento počet kroků je velmi malý. Celkový počet řešení potřebujících tento počet kroků dosahuje 15,54 %. Avšak i pro méně úspěšné varianty B a C lze nalézt řešení počítající s takovýmto počtem kroků. Procentuální zastoupení počtu takovýchto řešení postupně od varianty A po E je: 22, 6, 12, 15, 11 (zaokrouhлено na celá čísla). Nelze tedy jednoznačně porovnat vhodnost všech variant nalézání řešení podle nejmenšího počtu kroků. Nicméně průměrný počet kroků potřebných k řešení je pro všechny varianty velmi podobný. Dále zobrazené histogramy dokazují, že omezení počtu kroků na 15 se ukázalo být vhodným limitním stropem, protože se k tomuto číslu blíží jen velmi malý počet všech nalezených řešení.

Histogramy počtu podmínkových pravidel použitých při řešení jsou zachyceny na obr.

6.3. Zde je ukázáno, že minimální počet pravidel použitých na řešení neklesá pod hodnotu 8 - i když řešení využívající tento počet pravidel je minimum - a maximální počet pravidel je totožný s počtem pravidel generovaným pro přechodovou funkci. Procentuální zastoupení řešení využívající tento počet je pak pro varianty od A do E následující: 8, 25, 29, 8, 12 (zaokrouhлено na celá čísla). K těmto hodnotám je však nutno poznamenat, že mají malou vypovídající hodnotu o tom, kolik konfigurací okolí buňky kombinace daných podmínkových pravidel v přechodové funkci skutečně pokrývá. Nicméně i průměrné počty použitých pravidel a kumulace vysokých hodnot okolo těchto průměru svědčí o tom, že většinový podíl nalezených řešení používá počet pravidel blízký se k stanovenému limitnímu maximu, tj. počet podmínkových pravidel v chromozomu.

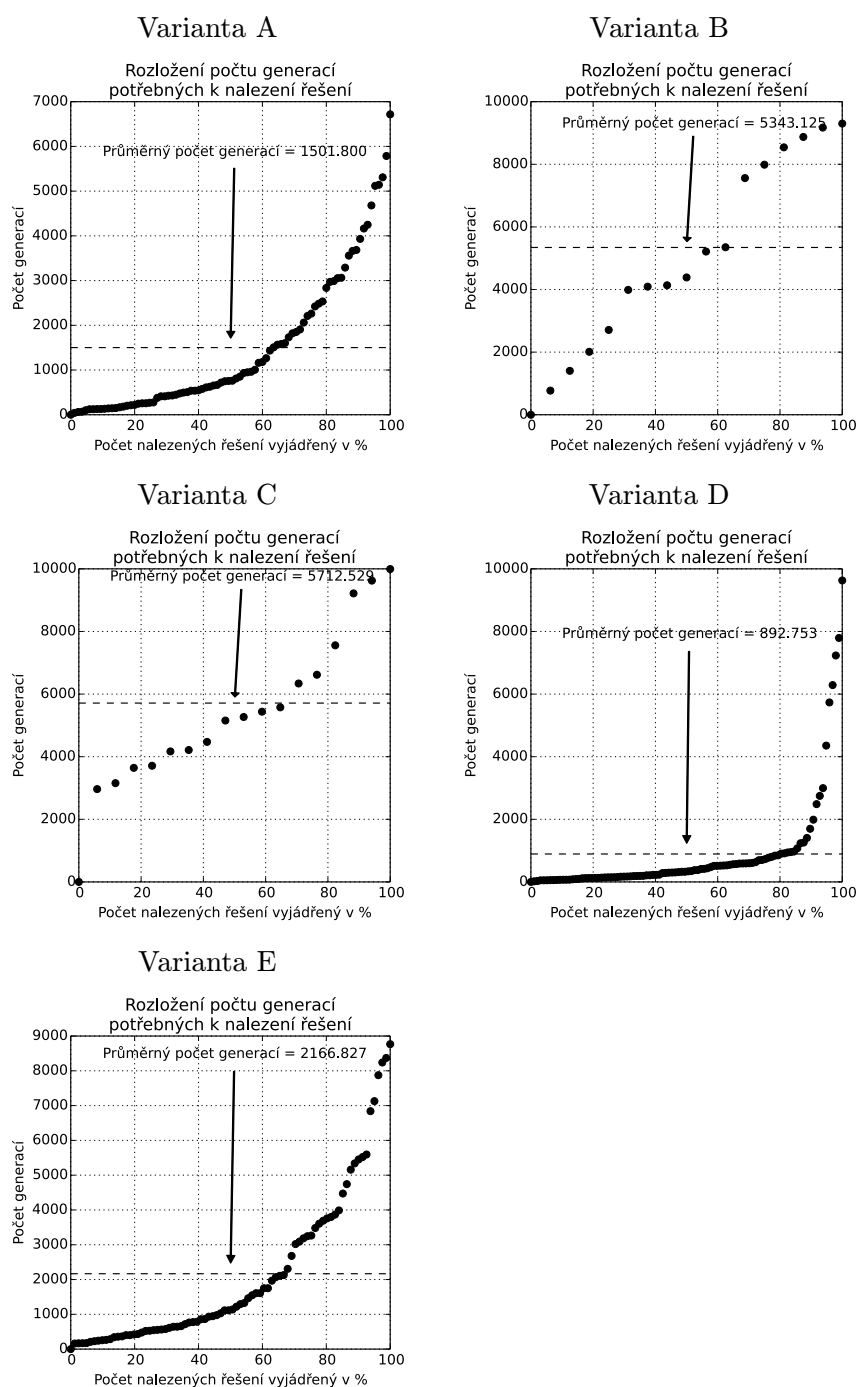
Na obrázku 6.4 je znázorněn průběh řešení všech možných kombinací vstupů pro variantu A. Červeně jsou označeny buňky, ve kterých se má po dokončení výpočtu nacházet výsledek. Šedá barva buňky značí bit ve stavu 1, bílá barva pak bit ve stavu 0. Přechodová funkce pro toto příkladové řešení je uvedena v tabulce 6.1. Na průběhu uvedených konfigurací CA je vidět, že některé z nich se dostávají do stabilní konfigurace už po druhém kroku - pokud není brána v úvahu konfigurace pro podmínku 0+0, která je stabilní po celý čas vývoje CA. S narůstajícím počtem kroků se zvyšuje počet konfigurací, které se stávají stabilními až nakonec v 5. kroku zůstává nestabilní pouze konfigurace pro podmínku 3+3. Jak je vidět, v přidavném dalším vygenerovaném kroku se právě tato konfigurace změní, a tím naruší splnění všech podmínek.

Index Pravidlo	0	1	2	3	4	5	6	7	8	Nový stav
1	0! =	1==	0 <=	0 <=	0==	0==	0==	1 >=	1 >=	1
2	0==	1*	0 >=	0 >=	0 >=	1==	0==	0! =	1! =	0
3	0*	0 >=	0==	1==	1 >=	1*	1*	1 >=	0 >=	1
4	1! =	0==	0 >=	0==	1 >=	1 >=	0 <=	1*	1! =	0
5	1==	1! =	1==	1 >=	0==	0*	0*	1==	1==	1
6	1 >=	1*	0==	1*	0*	0! =	1 >=	0! =	0! =	0
7	1*	0*	1 <=	0 <=	0 >=	1! =	0 <=	0 <=	0==	0
8	1! =	0 <=	1! =	1==	0==	1! =	0==	0*	1 >=	1
9	0==	0! =	0 >=	0==	1! =	1*	0==	0*	0 <=	1
10	1 <=	0 >=	0 <=	1==	1*	1! =	1 >=	1*	0 >=	1
11	1 <=	0==	1! =	1! =	1 <=	1*	0 >=	1==	1 >=	0
12	1 <=	1 >=	0 <=	1==	1 >=	0*	0 <=	0 >=	0! =	0

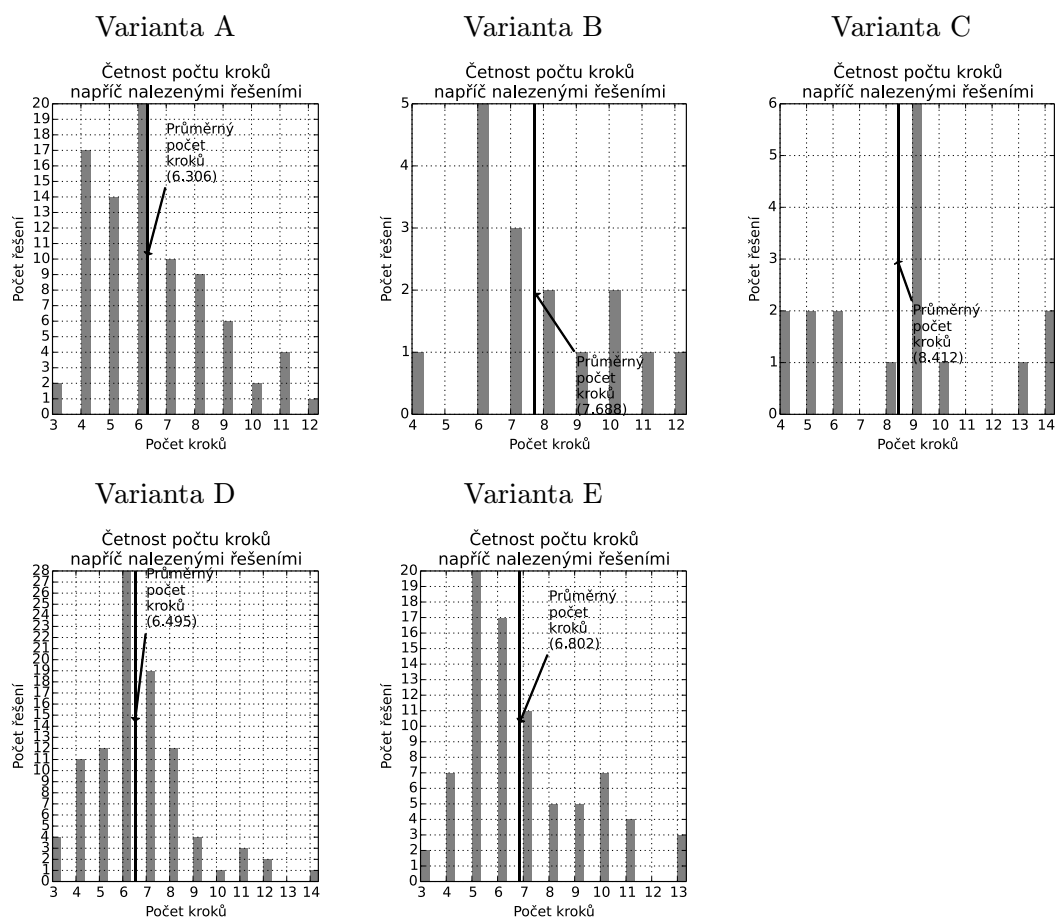
4	3	2
5	0	1
6	7	8

(a) Tabulka indexů okolí buňky

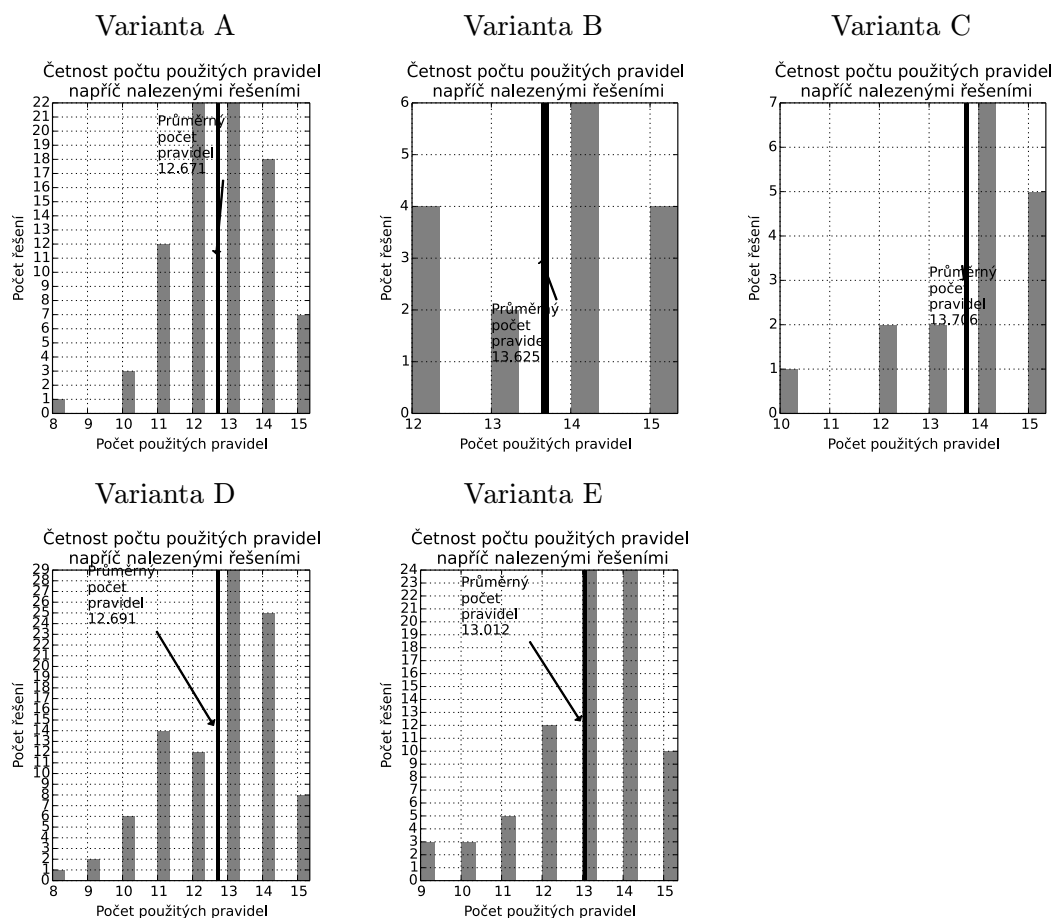
Tabulka 6.1: Přechodová tabulka pravidel pro příklad na obr. 6.4



Obrázek 6.1: Rozložení počtu generací potřebných k nalezení řešení



Obrázek 6.2: Rozložení počtu kroků ke splnění podmínek



Obrázek 6.3: Rozložení počtu pravidel použitých řešením

krok	0+0	1+0	2+0	3+0	0+1	1+1	2+1	3+1	0+2	1+2	2+2	3+2	0+3	1+3	2+3	3+3
0																
1																
2																
3																
4																
5																
+1																

Obrázek 6.4: Příklad sčítání 2 bitových operandů - varianta A - řešení o 5 krocích + jeden krok navíc

6.1.1 Příklady nalezených řešení

Na obrázku 6.5 je znázorněna část průběhu řešení 2 bitového sčítání pro variantu D. Automat zde generuje nad meze rostoucí strukturu vykazující jistou pravidelnost, která je nejlépe patrná ve čtvrtém kroku. Buňky obsahující výsledek jsou spočítány na „vrcholu“ jakéhosi trojúhelníku, který automat generuje a tudíž i v dalších krocích zůstanou stále. Obrázek 6.6 ukazuje další zajímavý průběh 2 bitového sčítání. Vstupní operandy jsou propagovány směrem doprava. Bariéra na pevnos stanovených okrajových buněk (není uvedena na obrázku) pak tuto propagaci zastaví díky čemuž je výsledek správný.

krok	1+0	2+0	3+0	1+1	2+1	3+1	2+2	3+2	3+3
0									
1									
2									
3									
4									

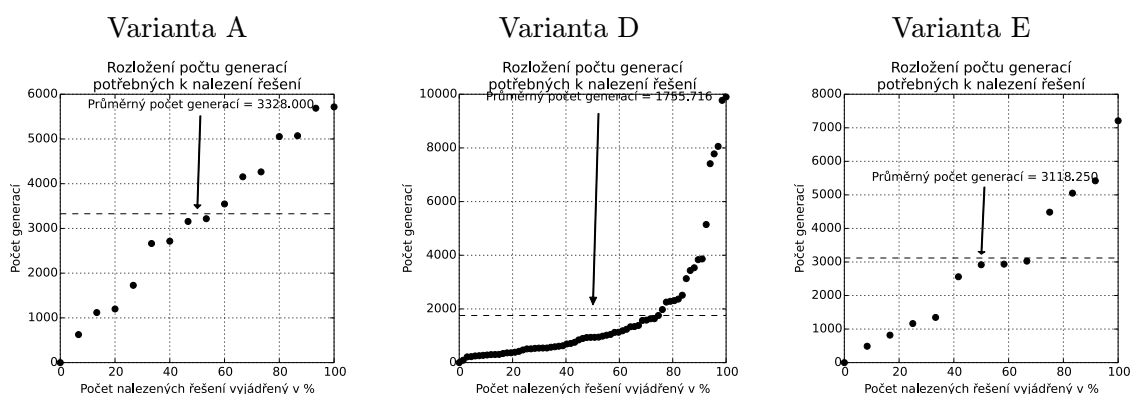
Obrázek 6.5: Příklad sčítání 2 bitových operandů pro vybrané podmínky - varianta D na rozšířené ploše automatu o $2 + v$ v kroku 4 o 3 (viz. sekce 6.5)

krok	0+0	1+0	2+0	3+0	0+1	1+1	2+1	3+1	0+2	1+2	2+2	3+2	0+3	1+3	2+3	3+3
0																
1																
2																
3																
4																
5																
6																

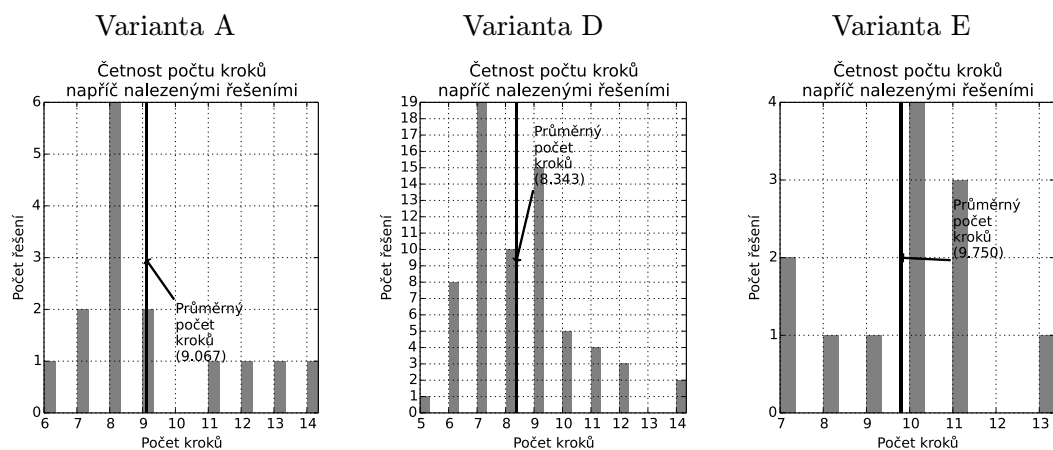
Obrázek 6.6: Příklad sčítání 2 bitových operandů - varianta E - řešení o 6 krocích

6.2 Sčítání 3 bitových operandů

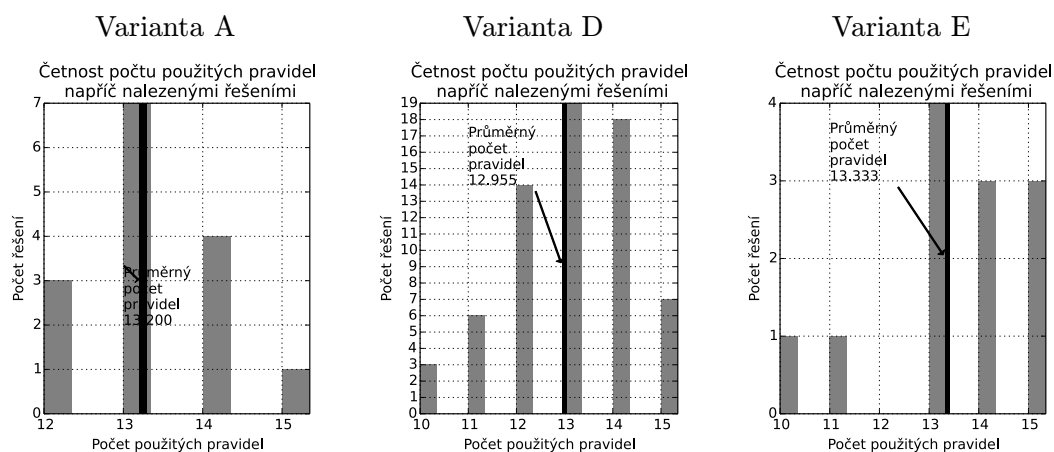
Jak se ukázalo, přidání jednoho bitu operandů a tím zčtyřnásobení počtu podmínek nutných pro přesné fungování CA mělo za důsledek rapidní úbytek počtu nalezených řešení. Pro variantu A je nalezeno pouze 15 řešení, což je 5,66 krát méně, než pro její 2 bitovou verzi. Obdobně je na tom varianta E, pro kterou bylo nalezeno 12 řešení, což je 6,75 krát méně, než v případě její předchozí. Výjimku tvoří varianta D, pro kterou bylo nalezeno 67 řešení, což je 69 % řešení nalezených pro 2 bitovou variantu D. Průměrný počet kroků vzrostl oproti 2 bitovému sčítání o přibližně 2,5 kroku a minimální počet kroků v průměru o 3 kroky. Maximální počet kroků vzrostl průměrně zhruba o 1,3 kroku. Na obr. 6.8 jsou zachyceny počty kroků a řešení, které daný počet kroků spotřebovaly. Z grafů je patrné, že stejně jako pro 2bitové sčítání se největší počty řešení vyskytují kolem průměrného počtu kroků. Maximální počet kroků využije ke splnění podmínek pouze asi 5 % z celkového počtu nalezených řešení. Minimální počet kroků se pohybuje v intervalu 5-7 kroků a procentuální zastoupení nalezených řešení využívajících tento počet kroků je pro varianty A,D,E následující: 20, 41, 16. Jedná se o podobná čísla jako se objevují v jejich 2 bitových verzích. Výjimku tvoří procentuální zastoupení u varianty D, což je způsobeno širším intervalem minimálních řešení. Po zúžení tohoto intervalu na 5-6 kroků je procentuální zastoupení pro variantu D 11, což je hodnota blízká se procentuálnímu zastoupení minimálních řešení u dvoubitového sčítání. Průměrný počet pravidel je pro 2 bitové i 3 bitové sčítání velmi podobný. Celkově se potřebný počet pravidel posouvá k maximu (15).



Obrázek 6.7: Rozložení počtu generací potřebných k nalezení řešení











































Obrázek 6.8: Rozložení počtu kroků ke splnění podmínek



Obrázek 6.9: Rozložení počtu pravidel použitých řešením

6.2.1 Příklady nalezených řešení

Na obr. 6.10 je znázorněn průběh výpočtu sčítání 3 bitových operandů pro variantu D. Jak je vidět aktivní buňky generují svislé sekvence dalších aktivních buněk, jsou které díky pevným okrajovým podmínkám vpravo „ohnuty“. Pokud už je ve stejném řádku jiná aktivní buňka, generovaná sekvence se naruší a díky čemuž vznikají nepravidelnosti, které po správném počtu kroků (12) dostanou automat do správné konfigurace. Obdobná situace je i v příkladu řešení pro variantu E na obrázku 6.11.

krok	1+0	7+0	3+1	1+7	3+3	3+6	3+7	7+7
0								
1								
2								
3								
4								

Obrázek 6.10: Příklad sčítání 3 bitových operandů pro vybrané podmínky - varianta D - první 4 kroky

krok	1+0	7+0	3+1	1+7	3+3	3+6	3+7	7+7
0								
1								
2								
3								
4								

Obrázek 6.11: Příklad sčítání 3 bitových operandů pro vybrané podmínky - varianta E - první 4 kroky

6.3 Násobení 2 bitových operandů

U násobení 2 bitových operandů jsou počty nalezených řešení u všech testovaných variant daleko méně úspěšné než u sčítání. Průměrná šance že bude nalezeno řešení je „jen“ 39,2 %, což je dáno více rozdílnými počty nalezených řešení, což lze vidět v tabulce 5.3. Průměrný počet generací k nalezení řešení vzrostl oproti variantám u sčítání na 3756,877, což je způsobeno menšími výkyvy počtu potřebných generací u jednotlivých variant. Výjimku tvoří pouze varianta D, která vykazuje největší poměr nalezených řešení ku počtu běhů (92:100). Porovnání situace u sčítání i násobení opět utvrzuje v tom, úspěšnost nalezení řešení v daném běhu je do značné míry ovlivněna volbou přechodových funkcí jedinců počáteční populace. Na obr. 6.12 je vidět rozložení počtu generací na počet jedinců vyjádřený v procentech. Opět je zde vidět závislost tvaru výsledné křivky, která vznikne propojením bodů, na celkovém počtu nalezených řešení. Podobnost průběhu křivek exponenciálně je ovšem menší. U všech variant je více než 50 % řešení nalezeno před dosažením průměrného počtu generací potřebných k nalezení řešení. Rozptyl průměrného počtu kroků potřebných k dosažení všech stanovených podmínek je 5-7 kroků, přičemž - stejně jako u sčítání - nejvyšší počty řešení se kumulují právě kolem tohoto intervalu. Minimální počet kroků potřebných pro řešení se pohybuje od 2 do 4, zatímco u sčítání byl tento interval menší. Počet řešení, která ke splnění všech podmínek potřebují minimální počet kroků je 84, což zastupuje 43,8 % všech nalezených řešení. Procentuálně je pro jednotlivé varianty od A po E zastoupení řešení využívajících minimální počet kroků následující: 11, 23, 36, 54, 36 (zaokrouhleno na celá čísla). Podrobnější údaje nabízí obrázek 6.13. Maximálního možného počtu kroků stejně jako u sčítání nevyužilo ani jedno nalezené řešení. Nejvyšší maximální počet kroků je u násobení také nižší než u sčítání. Co se týká počtu použitých pravidel přechodové funkce, situace je velmi podobná, jako u sčítání. Minimální počet pravidel použitých pro řešení je opět 8. Průměr se pohybuje od 11 do 13 pravidel, což je o něco méně, než u sčítání, a vysoké zastoupení řešení s počty pravidel kolem tohoto průměru (okolo spodní i horní hranice), zatímco u sčítání se značná část řešení nalezla od průměru výše. U sčítání byl celkový podíl

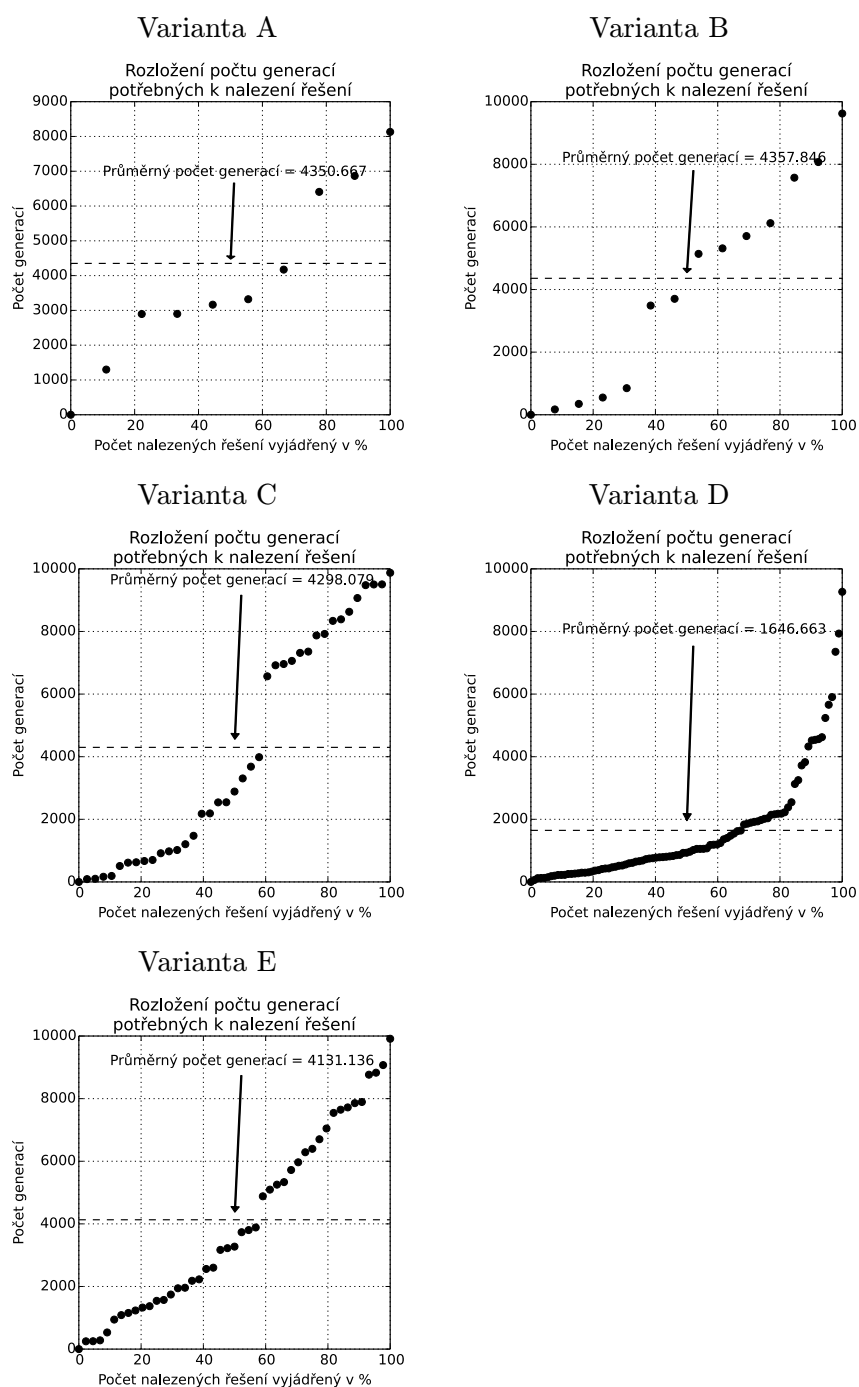
řešení využívajících maximální možný počet podmínkových pravidel 11,49 %; u násobení činí tento podíl pouhých 1,5 %. Na obrázku 6.15 je znázorněn průběh CA pro násobení ve varianty D o dvou krocích s jedním dodatečným dalším krokem. Právě v tomto dodatečném kroku je porušena podmínka $1 * 2$, a řešení tedy nesplňuje všechny podmínky.

Index Pravidlo	0	1	2	3	4	5	6	7	8	Nový stav
1	0==	0==	0 >=	1 >=	0*	0*	1 <=	1!=	1==	1
2	0==	1 <=	1 >=	0==	1!=	0 >=	1 >=	0!=	0==	1
3	1!=	1*	1 <=	0==	0 >=	0==	0!=	1 >=	0!=	1
4	1 >=	0!=	1!=	0 >=	0 >=	1 <=	1==	0 >=	1 >=	1
5	0 <=	0 <=	0!=	1 <=	1 <=	1 >=	0 >=	0*	1==	1
6	0==	0==	0*	1!=	0==	0==	0 >=	1 <=	0 >=	1
7	1!=	1 >=	0 >=	1==	1!=	0==	0 <=	0==	1!=	1
8	1!=	0 >=	1 >=	0 >=	1 >=	1 <=	1!=	1 <=	0!=	1
9	0 <=	1==	0!=	0*	1 <=	1==	1 <=	1*	0 >=	1
10	0*	1 >=	1*	0!=	1==	1 <=	1 <=	0 <=	0 <=	0

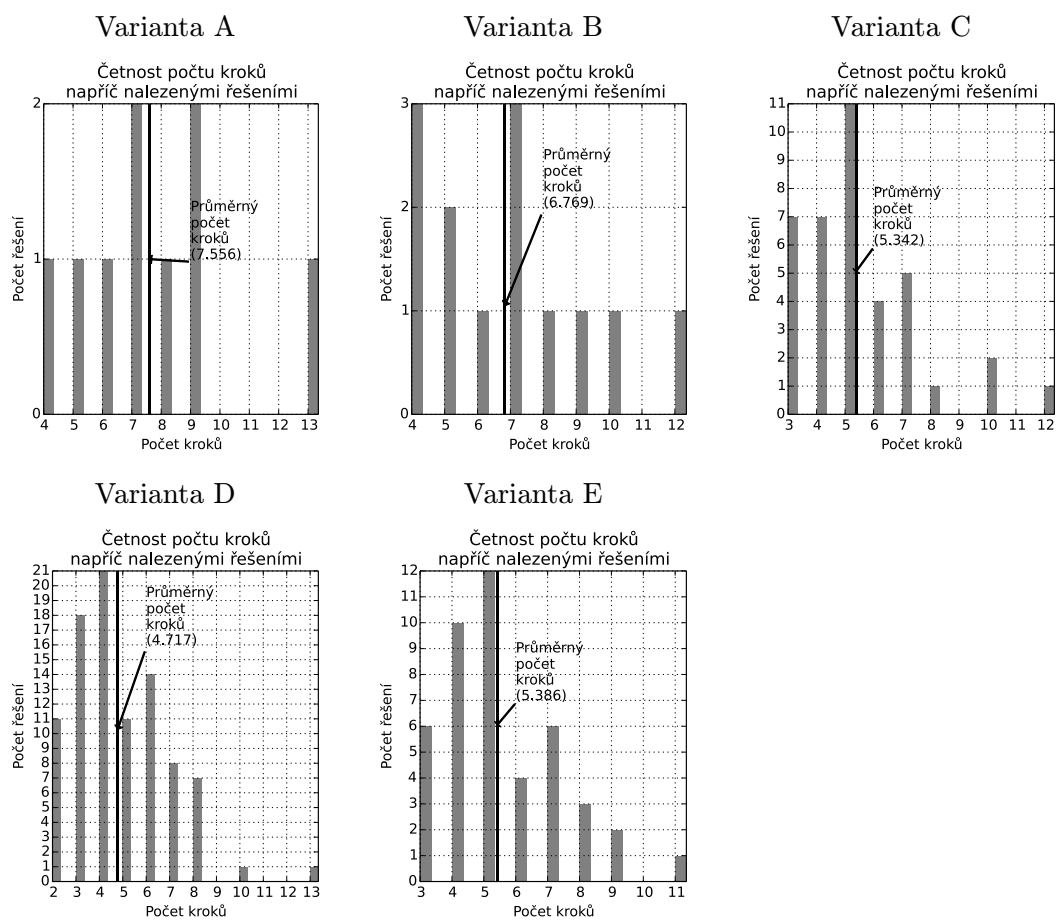
4	3	2
5	0	1
6	7	8

(a) Tabulka indexů okolí buňky

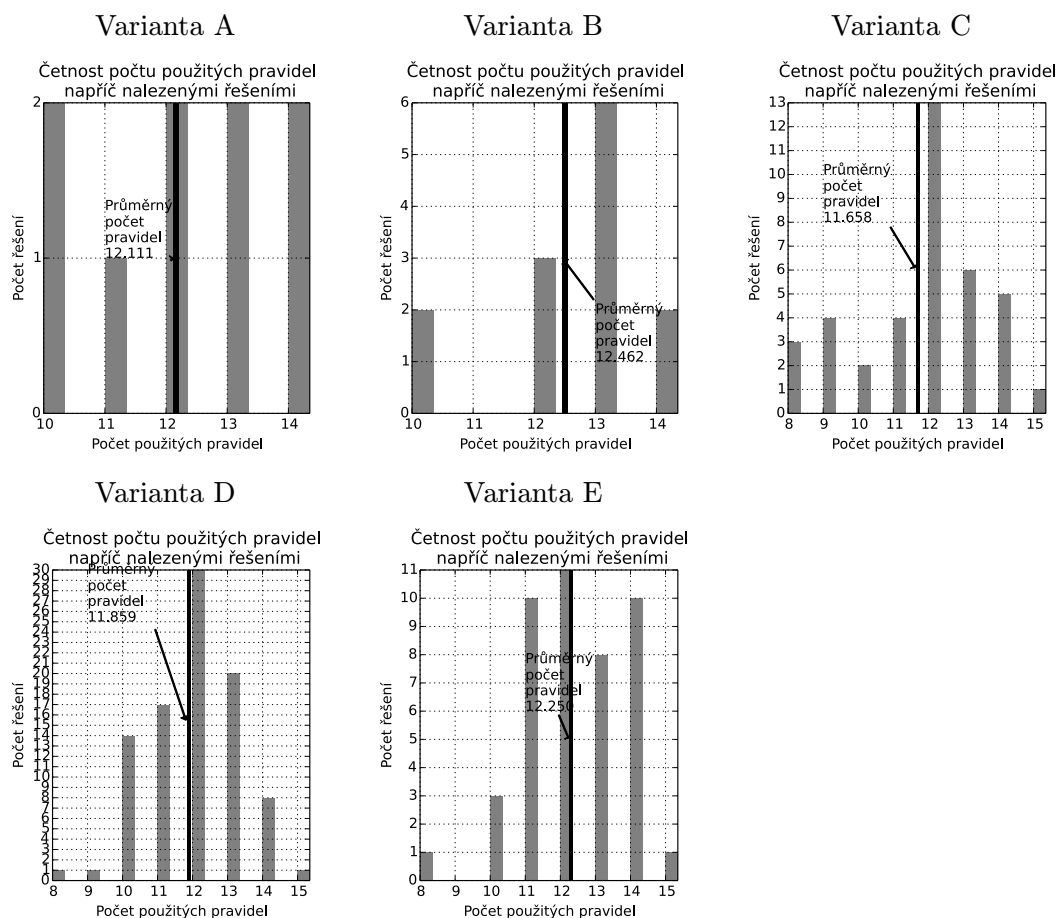
Tabulka 6.2: Přejchodová tabulka pravidel pro příklad na obr. 6.15



Obrázek 6.12: Rozložení počtu generací potřebných k nalezení řešení



Obrázek 6.13: Rozložení počtu kroků ke splnění podmínek



Obrázek 6.14: Rozložení počtu pravidel použitých řešením

krok	0*0	1*0	2*0	3*0	0*1	1*1	2*1	3*1	0*2	1*2	2*2	3*2	0*3	1*3	2*3	3*3
0																
1																
2																
+1																

Obrázek 6.15: Příklad násobení 2 bitových operandů - varianta D - řešení o 2 krocích + 1 krok

6.3.1 Příklady nalezených řešení

Obrázek 6.16 ukazuje část průběhu CA pro násobení 2 bitových operandů varianty C. Buňky nacházející se ve stavu 1 začíná generovat trojúhelníku podobnou strukturu. Jak je vidět pro libovolnou podmínku $n * 0$, případně $0 * n$, struktura je generována směrem dolů a proto neovlivní výsledek. Pokud však je více buněk ve stavu 1 vedle sebe - ve vhodné poloze, dochází k deformaci generované struktury a tím i nastavení správného výsledku. Obrázek 6.17 ukazuje odlišnou možnost spočítání řešení, kdy hned po prvním kroku jsou téměř všechny (v závislosti na podmínce) buňky, negovány a tím se jich většina dostává do stavu 1. V následujících krocích jsou pak buňky postupně nulovány.

krok	1*0	2*0	3*0	0*1	1*1	2*1	3*1	1*2	2*2	3*2	3*3
0											
1											
2											
3											
4											

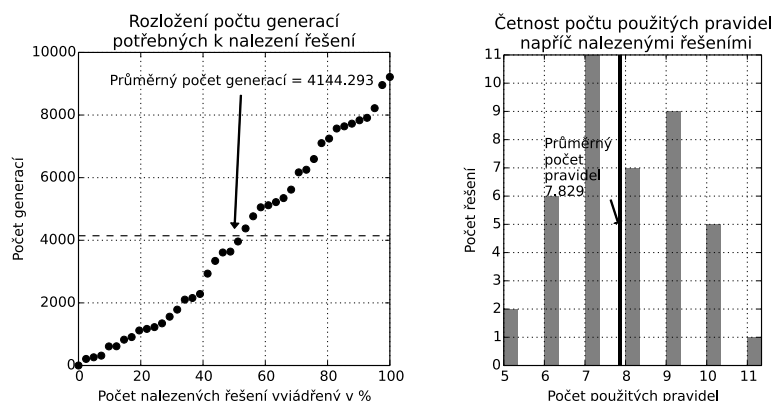
Obrázek 6.16: Příklad násobení 2 bitových operandů pro vybrané podmínky - varianta C na rozšířené ploše automatu o 1 (viz. sekce 6.5)

krok	0*0	1*0	2*0	3*0	0*1	1*1	2*1	3*1	0*2	1*2	2*2	3*2	0*3	1*3	2*3	3*3
0																
1																
2																
3																

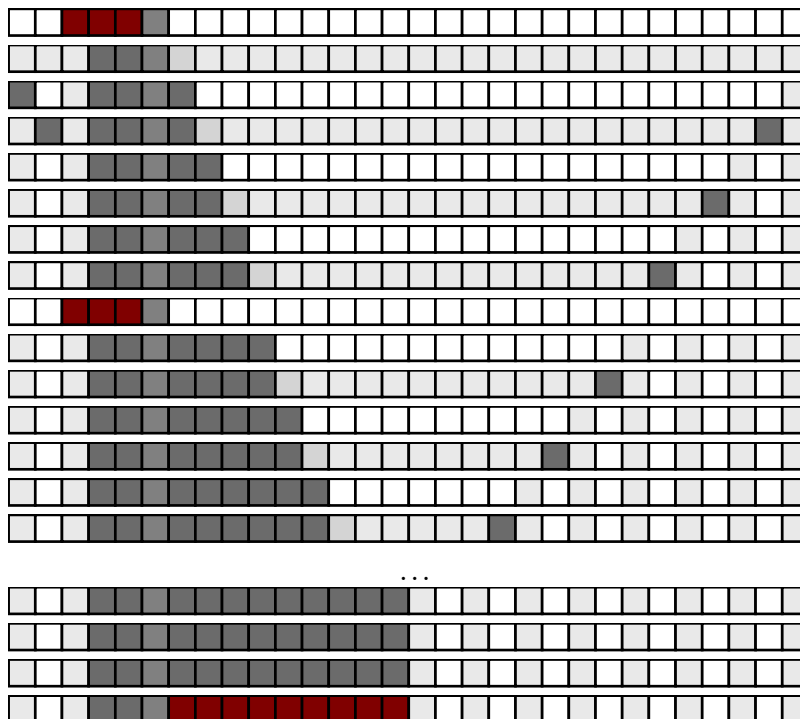
Obrázek 6.17: Příklad násobení 2 bitových operandů - varianta E - řešení o 3 krocích

6.4 N^2 ve vícestavovém 1D CA

Na obr. 6.18 je zobrazeno rozložení počtu potřebných generací pro nalezení řešení a počet použitých pravidel pro nalezení řešení. Z obrázku je patrné, že omezení počtu generovaných podmínkových pravidel na 12 se ukázalo být vhodnou volbou, protože tento počet nepoužívá žádné řešení a největší počty řešení se kumulují kolem průměru. Rozložení počtu generací potřebných k nalezení řešení vykazuje lineární průběh, což napovídá o tom, že to jak rychle bude nalezeno řešení je do značné míry ovlivněno vygenerování počáteční populace. Na obr. 6.19 je ukázán příklad průběhu mocniny pro číslo 3.



Obrázek 6.18: Statistické grafy pro druhou mocninu



Obrázek 6.19: Příklad průběhu mocniny

6.5 Mobilita nalezených řešení

Všechny výše uvedené varianty obvodů jsou počítány v poměrně malém prostoru buněk CA. Omezením prostoru na, kterém může být řešení počítáno, lze dosáhnout většího počtu nalezených řešení. Nevýhodou takto nalezených řešení je velká pravděpodobnost, že takovéto řešení může ke své správné funkci potřebovat právě přesně tento omezený prostor CA, protože ke své funkci využívá v některých případech neměnnost napevno určených okrajových buněk. Na druhou stranu se zase může stát, že ve větším prostoru CA by byla nalezena řešení, které by ve zvoleném prostoru nemohla fungovat. Přestože použitelnost řešení ve větším CA nebyla podmínkou k jejich nalezení, nalezená řešení byla dodatečně otestována na větších celulárních automatech. Tabulka 6.3 ukazuje kolik řešení z původních nalezených řešení je schopnost pracovat i ve zvětšeném automatu. Sloupec $+n$ určuje libovolně zvětšený automat o n buněk ve všech 4 směrech, kde n je celé číslo větší než 1. Hodnoty tohoto sloupce můžou být chápány takto obecně právě kvůli přesnému počtu kroků automatu, pro který je dán výsledek. Pokud chování automatu nezávisí na okrajových buňkách, je velmi pravděpodobné, že u většího automatu se už buňky nesoucí výsledek nestihnou do stanoveného počtu kroků nijak změnit. Zvětšení bylo testováno postupně od 1 do 7 a sloupec $+n$ shrnuje výsledky od zvětšení 2, protože, podle předpokladu, se již dále počet řešení neměnil. Do sloupce $+n$ byla samozřejmě počítána pouze ta řešení, která fungovala i při zvětšení 1. Z tabulky je patrná jistá úměra počtu nalezených řešení a počtu řešení fungujících ve větším automatu. U sčítání dvoubitových operandů je procentuální zastoupení fungujících řešení na větším automatu následující: 9, 41, 0, 0, 19, 58, 13, 58 (od A po E). Což v celkovém součtu dává 12,83 % ze všech nalezených řešení. U násobení byla tato situace výrazně lepší protože celkový počet fungujících řešení je více než dvojnásobný (35,20 %). Procentuální zastoupení pro násobení je u jednotlivých variant následující: 0, 15, 38, 63, 15, 40, 21, 13, 63. U sčítání 3 bitových čísel je nalezeno pouze jedno řešení (zastupuje asi 1 % všech řešení), které splňuje podmínky pro automat zvětšení o více než buňku v každém ze 4 směrů.

Varianta	Zvětšení		
	+0	+1	+n
A	85	8	8
B	16	0	0
C	17	0	0
D	97	19	19
E	81	11	11

(a) Tabulka zvětšení prostoru CA pro 2 bitové sčítání

Varianta	Zvětšení		
	+0	+1	+n
A	9	0	0
B	13	2	2
C	38	24	24
D	92	37	37
E	44	6	6

(b) Tabulka zvětšení prostoru CA pro 2 bitové násobení

Varianta	Zvětšení		
	+0	+1	+n
A	15	1	0
D	67	3	1
E	12	0	0

(c) Tabulka zvětšení prostoru CA pro 3 bitové sčítání

Tabulka 6.3: Tabulka zvětšení prostoru CA

Zvětšení $+0$ určuje počet nalezených řešení v automatu s původní velikostí. Zvětšení $+1$ pak udává počet řešení fungujících na automatu s prostorem o 1 větším do všech 4 směrů. Zvětšení $+n$ udává počet řešení fungujících na automatech s prostorem větším o n v každém směru, kde n je celé číslo větší než 1.

6.6 Shrnutí dosažených výsledků

Nalezené výsledky dokazují, že zvolené parametry genetického algoritmu umožňují nalézt řešení v obecně krátkém počtu generací. Porovnání výsledků u sčítání a násobení ukazuje, že u násobení průměrný počet potřebných generací pro nalezení řešení téměř dvojnásobný. Zatím co u sčítání byl počet nalezených řešení vyšší, u násobení se za cenu menšího počtu nalezených řešení podařilo nalézt více než dvojnásobně řešení s minimálním počtem kroků. Navíc se u násobení podařilo pro variantu D nalézt 11 řešení (asi 12 % všech řešení pro variantu D) využívající pouze 2 kroky, což se u sčítání pro žádnou z variant nepodařilo. Průměrný počet kroků u násobení je menší a i celkově je počet potřebných kroků pro násobení menší než pro sčítání. U průměrného počtu použitých pravidel je situace podobná. Všechny tyto fakty nasvědčují tomu, že i když zvolená strategie pro selekci nových kandidátních řešení pracuje podle požadavků, tj. nalézt co nejrychleji a co nejvíce řešení, Při větším počtu generací jsou nalezené výsledky kvalitnější, co se týká nároků na počet kroků a počtu použitých pravidel. Zatímco u sčítání se pro vybrané varianty podařilo najít i řešení pracující s 3 bitovými operandy, u násobení nebylo takovéto řešení nalezeno pro žádnou z variant. Evoluční algoritmus s parametry, které pro něj byly zvoleny, v tomto případě selhal a otázkou zůstává, zda by jinak zvolené parametry mohly přinést alespoň nějaký úspěch. Stejně tak se nepodařilo nalézt příliš dobré výsledky při hledání mocniny. I přes poměrně slušnou úspěšnost hledání řešení mocniny pro maximální hodnotu, navýšení této hodnoty vedlo k absolutnímu selhání. Důvodem je pravděpodobně nevhodnost zvoleného algoritmu vyhodnocování GA pro tento experiment.

Dodatečný test řešení na jejich správnou funkci ve větším automatu ukázal, že i přestože toto nebylo podmínkou, značné procento tuto podmínku splňuje. Překvapující je fakt, že větší zastoupení měla takováto řešení u násobení, pro které bylo nalezeno celkově méně řešení.

Kapitola 7

Závěr

V rámci práce byly probrány možnosti celulárních automatů. Na několika příkladech byla prozkoumána schopnost univerzálního výpočtu spolu s naznačením dalších možností ne-univerzálních výpočtů. Detailně byly popsány možnosti použití evolučních algoritmů pro vývoj celulárního automatu. Bylo popsáno problémy vzniklé při použití tohoto procesu. Následně byl popsán zvolený způsob vyhodnocování genetického algoritmu, který byl vybrán jako vhodný evoluční algoritmus pro tuto práci. Popsány byly parametry jeho nastavení s odůvodněním proč tomu tak bylo.

Pro vybrané obvody byly zvoleny varianty zakódování jejich vstupů a výstupů do CA. Pro každou z těchto variant byla provedena sada testů, která měla určit do jaké míry je varianta vhodná pro hledání řešení příslušného obvodu. U nalezených řešení bylo provedeno srovnání jejich náročnosti na nalezení a náročnosti na spotřebované zdroje.

Na nalezených řešeních se ukázalo, že navržený algoritmus vyhodnocování je schopný nalézt s velkou pravděpodobností řešení a to za použití relativně malého počtu generací. Omezením u takto nalezených řešení by mohla být nutnost explicitně dodat informaci přesném počtu kroků, který je potřeba k vyhodnocení řešení. Nicméně zvolená metoda vyhodnocování GA díky tomuto poskytuje celou škálu řešení pracujících s různým počtem kroků. Dalším omezením by v jistém směru mohla být skutečnost, že většina nalezených řešení není přenositelná na větší automat. V sekci o mobilitě řešení je ale demostrováno, že některá z nalezených řešení přenositelná jsou.

Navázání na tuto práci může být zaměřeno na hledání lepších parametrů, které by pro zvolený algoritmus vyhodnocování našly lepší výsledky. Díky informacím poskytnutým v kapitole 6 je možné upravit parametry způsobem tak, aby byly v jeho průběhu omezeny extrémní situace (jako např. neúměrně velký limit generací, zbytečně velký maximální počet kroků a podobně). Pro hledání nových obvodů realizovatelných na CA můžou zvolené varianty u vybraných obvodů, pro které byla prokázána vysoká úspěšnost, poskytnout inspiraci pro volbu zakódování vstupů a výstupů obvodu.

Další možnost výzkumu spočívá v přidání omezujících podmínek pro hledané řešení, jako například požadavek na přenositelnost řešení na větší automat zmíněný výše nebo požadavek na neměnnost buněk s výsledkem v průběhu dalšího vývoje CA.

Literatura

- [1] Úžasná Zeměplocha, Talpress, 2006, ISBN 9788071972778.
- [2] Bidlo, M.: Evolution of Computational Processes in Uniform Cellular Automata. *Phys. D*, ročník 2, č. 1-3, Říjen 2014: s. 120–149, ISSN 0167-2789.
- [3] Burks, A. W.: *Von Neumann's self-reproducing automata*. University of Michigan, 1969.
- [4] Callahan, P.: Stable glider reflector and Herschel tracks. 2011.
URL <<http://www.radicleye.com/lifepage/patterns/p1/p1.html>>
- [5] Gajardo, A.; Moreira, A.; Goles, E.: Complexity of Langton's ant. *Discrete Applied Mathematics*, ročník 117, č. 1–3, 2002: s. 41 – 50, ISSN 0166-218X.
- [6] Hoekstra, A.; Kroc, J.; Sloot, P.: *Simulating Complex Systems by Cellular Automata*. Springer Complexity: Understanding Complex Systems, Springer, 2010, ISBN 9783642122026.
- [7] Hynek, J.: *Genetické algoritmy a genetické programování*. 2008, ISBN 9788024760575.
- [8] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. Edícia vysokoškolských učebníc / Slovenská technická univerzita, Slovenská technická univerzita, 2000, ISBN 9788022713771.
- [9] Langton, C. G.: Studying Artificial Life with Cellular Automata. *Phys. D*, ročník 2, č. 1-3, Říjen 1986: s. 120–149, ISSN 0167-2789.
- [10] Langton, C. G.: Studying Artificial Life with Cellular Automata. *Phys. D*, ročník 2, č. 1-3, oct 1986: s. 120–149, ISSN 0167-2789.
- [11] Lindgren, K.; Nordahl, M. G.: Universal computation in simple one-dimensional cellular automata. *Complex Systems*, ročník 4, č. 3, 1990: s. 299–318.
- [12] Mazoyer, J.: *Cellular Automata: A Parallel Model*. Mathematics and Its Applications, Springer, 1999, ISBN 9780792354932.
- [13] Mitchell, M.: *An Introduction to Genetic Algorithms*. A Bradford book, MIT Press, 1998, ISBN 9780262631853.
URL <<http://books.google.ca/books?id=0eznlz0TF-IC>>
- [14] Neary, T.: *Small universal Turing machines*. Diplomová práce, National University of Ireland, Maymooth, Ireland, 2008.

- [15] Petraglio, E.; Tempesti, G.; Henry, J.-M.: Arithmetic Operations with Self-Replicating Loops. In *Collision-Based Computing*, editace A. Adamatzky, Springer London, 2002, ISBN 978-1-85233-540-3, s. 469–490.
- [16] Sarkar, P.: A brief history of cellular automata. 2000.
- [17] Schwartz, J. T.; Society, A.: *Mathematical Aspects of Computer Science*. American Mathematical Society. Proceedings of Symposia in Applied Mathematics, American Mathematical Soc., ISBN 9780821867280.
- [18] Sipper, M.: *Evolution of parallel cellular machines: the cellular programming approach*. Lecture notes in computer science, Springer, 1997, ISBN 9783540626138.
- [19] Thompson, A.: An evolved circuit, intrinsic in silicon, entwined with physics. In *Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science*, ročník 1259, editace T. Higuchi; M. Iwata; W. Liu, Springer Berlin Heidelberg, 1997, ISBN 978-3-540-63173-6, s. 390–405.
- [20] Wolfram, S.: *A New Kind of Science*. Wolfram Media, 2002, ISBN 9780713991161.

Seznam příloh

Manuál pro kolekci testovacích skriptů	56
--------------------------------------------------	----

Příloha A

Manuál pro kolekci testovacích skriptů

A.1 Obsah CD

CD obsahuje archiv `ca.tar.gz`, který po rozbalení obsahuje zdrojové kódy knihovny a pomocných skriptů vyhotovených v rámci práce, a archiv `tex.tar.gz`, který obsahuje zdrojové soubory \LaTeX této práce.

A.2 Požadavky na běh testů

Pro kompilaci knihovny jsou potřeba následující závislosti:

- `cmake` (≥ 2.6)
- kompilátor jazyka C
- podpora PThreads v jádře
- `PythonLibs2` ($\geq 2.7.5$)
- `glibc` ($\geq 1.2.1$)

Pro běh testů jsou potřeba následující závislosti:

- `python2` ($\geq 2.7.5$)

A.3 Kompilace

Knihovna se zkompiluje příkazem `cmake + make`

```
$ mkdir build; cd build
$ cmake <path.to.source.root.dir>
$ make
```

Možnost instalace není možná, lze však všechny potřebné soubory nakopírovat do jednoho adresáře

```
$ make copy DST_DIR=<path>
```

A.4 Spuštění experimentu

Experiment lze spustit skriptem `test.py`. Skript vyžaduje následující parametry:

```
--s_limit <int> - maximální možný počet kroků na řešení  
--p_limit <int> - maximální možný počet kroků na běh  
--pop_size <int> - velikost populace  
--r_count <int> - počet pravidel přechodové funkce  
--form <string> - typ experimentu  
--file <path> - soubor s výsledky
```

Poslední povinný poziční parametr udává kolik nezávislých běhů, který bude provedeno. Soubor s výsledky není přepsán pokud existuje, lze tedy spustit experiment vícekrát a tím dostat požadovaný počet běhů. Na výsledný soubor jednoho typu experimentu nelze spustit experiment jiného typu.

Pro seznam všech možných typů experimentu lze spustit skript s parametrem `--help`.

Příklad spuštění:

```
./test.py --s_limit 15 --p_limit 10000 --pop_size 20 --r_count 15 --form add2x2_4  
--file add2x2_4.10000.out
```

Typy experimentů jsou v adresáři `cond_forms`. Jedná se o jednoduché skripty/konfigurační moduly v jazyce python

A.5 Popis konfiguračního souboru experimentu

DNC = -1 - stav buňky v podmínce se nebude kontrolovat (do not care)

X = DNC - pomocná proměnná

A = "A"- pomocné proměnné pro substituce,

B = "B"

C = "C"

Pole `init` určuje vstupní konfiguraci CA. Řetězcové konstanty slouží k nahrazení konkrétními hodnotami. Více bude popsáno níže.

```
init = [[0,0,0,0,0,0,0],  
        [0,0,0,0,0,0,0],  
        [0,0,0,A,0,0,0],  
        [0,0,A,0,B,0,0],  
        [0,0,0,B,0,0,0],  
        [0,0,0,0,0,0,0],  
        [0,0,0,0,0,0,0]]
```

Pole `end` podobně jako pole `init` slouží k určení výstupní požadované konfigurace. Hodnoty X specifikují že na stavu dané buňky nebude záležet. Hodnoty 0 na okrajích pole jsou pro odlišení okrajových buněk CA, které se vývoje konfigurace automatu neúčastní.

```
end = [[0,0,0,0,0,0,0],  
       [0,X,X,X,X,X,0],  
       [0,X,X,X,X,X,0],  
       [0,X,X,C,X,X,0],  
       [0,X,C,X,X,X,0],  
       [0,C,X,X,X,X,0],  
       [0,0,0,0,0,0,0]]
```

CONDS - proměnná uchovávající podmínky experimentu ve formátu `[(init_0, end_0), ... , (init_n, end_n)]`, kde `init_x` a `end_x` jsou pole `init` a `end` vyplněné konkrétními hodnotami.

Funkce `cond_utils.make_conds` slouží k vytvoření pole pro proměnou `CONDS`. Funkce má následující signaturu:

```
cond_utils.make_conds(init=<init_array_form>,
                      inputs=[{'<subst_val1>':[int]}, ...],
                      end=<end_array_form>,
                      outputs=[{'<subst_val1>':[int]}, ...],
                      dnc=<Do Not Care int state>)
```

Konkrétní příklad zavolání funkce:

```
cond_utils.make_conds(init=init, # viz. pole init vyse
                      inputs=[{'A':[0,1], 'B':[1,1]}, {'A':[1,1],
'B':[1,1]}],
                      end=end, # viz. pole end vyse
                      outputs=[{'C':[1,0,1]}, {'C':[1,1]}],
                      dnc=DNC)
```

Toto zavolání vytvoří 2 podmínky tak že substituuje řetězcové proměnné A, B, C v polích výše konkrétními hodnoty ze slovníku `inputs` a `outputs`. Substituce probíhá po řádcích zleva doprava.

`NBHD_COUNT` - proměnná uchovávající počet buněk okolí -1

`NBHD_INDEXES` - proměnná uchovávající indexy okolních buněk středové buňky v matici CA ve formátu:

```
[[x-cord1, x-cord2], [y-cord1, y-cord2]]
```

`MAT_EDGE` - proměnná určující zda bude použito pomocných okrajových buněk v CA - 1 = Ano.

`MAT_WIDTH` - šířka matice CA, bez započítání pomocných buněk

`MAT_HEIGHT` - výška matice CA bez započítání pomocných buněk

`MAX_STATE` - maximální stav buňky, který může buňka nabývat. Číslování stavů začíná od 0.

`CONDS_TYPE` - určuje zda budou použita přímá nebo podmíněná kontrola cílové konfigurace CA

A.6 Vygenerování průběhu řešení

Skriptem `sol_runner.py` lze vygenerovat textový průběh vývoje CA pro všechny podmínky. Parametry skriptu jsou následující:

`--file` - soubor vytvořený skriptem `test.py`

`--extra_steps` - dodatečný počet kroků nad počet kroků pro nalezené řešení

`--dst_dir` - cílový adresář pro generování výsledků

skript vygeneruje soubory s postfixem `_sol[x].ca` a `_sol[x].rules` - kde `x` je index nalezeného řešení.

Soubor s koncovkou `.ca` obsahuje vygenerované průběh pro všechny podmínky experimentu. Soubor s koncovkou `.rules` obsahuje pravidla pro nalezené řešení v textovém formátu, kde každé pravidlo je na samostatném řádku. První sloupec určuje buňku, na kterou je pravidlo aplikováno, ostatní sloupce určují buňky podle pořadí souřadnic v poli `NBHD_INDEXES`. Poslední sloupec určuje výsledný nový stav buňky.

Příklad spuštění:

```
./sol_runner.py --file add2x2_4.10000.out --extra_steps 1 --dst_dir runned
```

A.7 Vygenerování statistik pro výsledky běhů

Skript `print_stats.py` slouží ke generování statistik výsledků běhů. Jako vstupní parametr, přijímá jméno souboru, generovaného skriptem `test.py`. Pokud bude nalezen modul `matplotlib`, skript rovněž vygeneruje k zobrazeným statistikám i detailní grafy.

Příklad spuštění:

```
./print_stats.py add2x2_4_10000.out
```