



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

**STATISTICKÉ OVĚŘOVÁNÍ MODELŮ PŘIBLIŽNÝCH
VÝPOČETNÍCH SYSTÉMŮ**

STATISTICAL MODEL CHECKING OF APPROXIMATE COMPUTING SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL BLAŽEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOSEF STRNADEL, Ph.D.

BRNO 2024

Zadání bakalářské práce



Ústav: Ústav počítačových systémů (UPSY) 150953
Student: **Blažek Michal**
Program: Informační technologie
Název: **Statistické ověřování modelů přibližných výpočetních systémů**
Kategorie: Modelování a simulace
Akademický rok: 2023/24

Zadání:

1. Proveďte rešerši v oblastech statistického ověřování modelů (angl. Statistical Model Checking, SMC) a přibližného počítání (angl. Approximate Computing, AC).
2. Zvolte vhodné SMC prostředky pro modelování a analýzu AC systémů, jejich vlastnosti a dopadů na jejich okolí.
3. Vytvořte, popř. vygenerujte, modely vhodně vybraných zástupců zvolené třídy AC systémů (např. algoritmů či obvodů), ověřte jejich vlastnosti v různých podmínkách pomocí SMC prostředků a srovnajte je s vlastnostmi "přesných" variant těchto systémů.
4. Zhodnoťte řešení (modely, ověření, srovnání) a výsledky pomocí něj dosažené; identifikujte silné a slabé stránky řešení, navrhněte možné směry jeho úprav/vylepšení a rozveďte ty, které považujete za nejperspektivnější.

Literatura:

- Mittal, S.: A Survey of Techniques for Approximate Computing. ACM Computing Surveys, Vol. 48, No. 4, 2016, 33 p. DOI: 10.1145/2893356.
- David, A., Larsen, K., Legay, A., Mikučionis, M., Poulsen, D.: Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer. Springer Berlin Heidelberg, 2015, pp. 397 - 415, Vol. 17, No. 4. ISSN 1433-2779, DOI: 10.1007/s10009-014-0361-y.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání, návrh modelu jednoduchého approximativního systému.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**

Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.

Datum zadání: 1.11.2023

Termín pro odevzdání: 9.5.2024

Datum schválení: 30.10.2023

Abstrakt

Tato bakalářská práce se zaměřuje na statistické ověřování modelů přibližných násobiček. Zejména se zabývá srovnáním vlastností násobiček při generování jejich vstupních hodnot podle různých pravděpodobnostních rozdělení. Součástí práce je převod modelů násobiček z knihovny EvoApproxLib do modelů v prostředí UPPAAL. Vytvořené modely jsou poté simulovány s ohledem na vybrané hodnotící metriky, jako např. pravděpodobnost chyby, průměrná absolutní chyba aj.

Ze získaných výsledků lze usuzovat, že použitím vhodné approximační násobičky pro provádění výpočtů v rámci konkrétní aplikace je možné docílit menší chyby ve výpočtech. Výsledky by proto mohly mít další uplatnění v oblasti přibližných výpočetních systémů.

Abstract

This bachelor's thesis focuses on the simulation of models of approximate multipliers. The main aim of the thesis is comparing selected properties of multipliers in an application-specific scope of input values. The thesis includes the conversion of multiplier models from the EvoApproxLib library into models used in the UPPAAL environment. These models are then simulated while monitoring their selected evaluation metrics such as error probability, mean absolute error, etc.

From the obtained results, one can conclude that using a suitable approximate multiplier in a specific context can have a positive effect on the error in calculations. The results could therefore have further applications in the field of approximate computing systems.

Klíčová slova

modelování, simulace, aproximace, formální verifikace, statistické ověřování modelů, násobičky, approximační násobičky, stochastické časové automaty, UPPAAL, Python

Keywords

modelling, simulation, approximation, formal verification, statistical model checking, multipliers, approximate multipliers, stochastic timed automata, UPPAAL, Python

Citace

BLAŽEK, Michal. *Statistické ověřování modelů přibližných výpočetních systémů*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Statistické ověřování modelů přibližných výpočetních systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Blažek

5. května 2024

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Josefу Strnadelovi, Ph.D. za zadání zajímavého tématu a následnou pomoc a cenné rady při jeho řešení. Dále bych chtěl poděkovat své rodině a přátelům za jejich neutuchající podporu v průběhu celého studia.

Obsah

1	Úvod	5
2	Přibližné výpočetní systémy	6
2.1	Využitelnost a omezení přibližných výpočetních systémů	6
2.2	Aproximační techniky	8
2.3	Hodnotící metriky	10
2.4	Aproximační násobičky	12
3	Statistické ověřování modelů	18
3.1	Ověřování modelů	20
3.2	Úvod do statistického ověřování modelů	22
3.3	Časové automaty	23
3.4	Modelovací nástroj UPPAAL	25
4	Návrh a implementace řešení	34
4.1	Popis systému v nástroji UPPAAL	35
4.2	Převod modelů z jazyka Verilog do modelů v nástroji UPPAAL	39
4.3	Zvolené aplikace	41
4.4	Simulační dotazy	42
5	Výsledky experimentů	45
6	Diskuze škálovatelnosti	51
7	Závěr	54
Literatura		55
A	Obsah paměťového média	58
B	Návod na zprovoznění	59
C	Pravděpodobnostní grafy při odhadu max. nebo min. hodnoty	61
D	Pravděpodobnostní rozdělení při generování vstupů korespondující se zvolenými aplikacemi	63
E	Výsledky ověřovacích simulačních dotazů	71
F	Podrobné výsledky experimentů	74

Seznam obrázků

2.1	Příklady výstupů approximačních prahovacích algoritmů. Převzato z [13]	7
2.2	Přehled approximačních technik používaných ve strojovém učení. Převzato z [12]	8
2.3	Přibližná sčítacka lower-part-or s m nepřesnými bity. Převzato z [8]	10
2.4	Binární násobička 4x4 bity	12
2.5	Příklad Wallace-Tree násobičky. Převzato z [27]	13
2.6	(a) Příklad oříznutí DSM; (b) Příklad oříznutí SSM. Převzato z [27]	14
2.7	(a) Přibližná násobička 2x2 bity (b) Přesná násobička 2x2 bity	15
2.8	Tvorba násobiček z menších bloků. Převzato z [15]	15
2.9	Příklad rozdělení matice částečných součinů na skupiny s různou úrovní aproximace. Převzato z [27]	16
2.10	Příklad násobičky typu Broken-Array. Převzato z [17]	16
3.1	Rozdělení přístupů k verifikaci programů	18
3.2	Hierarchie kvantitativních automatů, převzato z [11]	19
3.3	V-model popisující fáze vývoje počítačového systému	20
3.4	Schéma principu ověřování modelů, převzato z [2]	21
3.5	Příklady stochastických časových automatů, převzato z [7]	24
3.6	Rozdělení dosažitelnosti lokace END, převzato z [7]	24
3.7	Jednoduchý model lampy. Převzato z [4]	26
3.8	Symbolický simulátor v programu UPPAAL	27
3.9	Verifikátor programu UPPAAL	28
3.10	Běhové formule v nástroji UPPAAL. Převzato z [4]	30
3.11	Vizualizace hodnot při simulaci, nástroj UPPAAL	32
3.12	Grafy pravděpodobnostního rozdělení při odhadu maximální hodnoty proměnné v nástroji UPPAAL	33
4.1	Schéma systému porovnávání výstupů přesné a přibližné násobičky	35
4.2	Časový automat reprezentující model přesné násobičky	36
4.3	Časový automat reprezentující model generátoru pseudonáhodných veličin .	36
4.4	Časový automat reprezentující model kontroloru rovnosti výsledků	37
4.5	Časový automat reprezentující model zajišťující synchronizaci vstupních hodnot	37
4.6	Časový automat reprezentující model logického hradla	38
4.7	Schéma fungování skriptu <code>parse.py</code>	40
4.8	Vytvoření instance logického hradla v nástroji UPPAAL z předlohy ve Verilogu	40
4.9	Aktualizace výstupního bitu logického hradla	41
4.10	Příklad výstupu dotazu ověřujícího hodnoty vygenerovaných vstupů	43

5.1	Pravděpodobnost chyby vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot	46
5.2	Porovnání pravděpodobnosti chyby sledovaných násobiček v rámci vybraných rozdělení	47
5.3	Porovnání průměrné absolutní chyby sledovaných násobiček v rámci vybraných rozdělení	47
5.4	Průměrná absolutní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot	48
5.5	Průměrná relativní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot	49
5.6	Porovnání průměrné relativní chyby sledovaných násobiček v rámci vybraných rozdělení	49
5.7	Nejhorší absolutní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot	50
6.1	Paměťová náročnost dle počtu vstupních bitů	52
6.2	Časová náročnost simulací v závislosti na počtu logických hradel	53
C.1	Příklad grafu rozdělení pravděpodobnosti	61
C.2	Příklad grafu rozdělení distribuční pravděpodobnosti	61
C.3	Příklad grafu konfidenčních intervalů distribuční funkce	62
C.4	Příklad histogramu frekvencí výskytu jednotlivých hodnot při simulačních bězích	62
D.1	(a) a (b) PDF násobených dvojic rasterizačního algoritmu Ellipse Mid-Point, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů	64
D.2	(a) a (b) PDF násobených dvojic Bresenhamova rasterizačního algoritmu, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů	65
D.3	(a) a (b) PDF násobených dvojic v algoritmu Pritchardovo síto, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů	66
D.4	(a) a (b) PDF násobených dvojic v algoritmu výpočtu celočíselné odmocniny, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů	67
D.5	(a) a (b) PDF násobených dvojic v algoritmu Circle Point to Point, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů	68
D.6	(a) a (b) PDF násobených dvojic v algoritmu AKS, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů	69
D.7	(a) a (b) PDF násobených dvojic v algoritmu ElGamal, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů	70
E.1	Rozdělení uni_uni	71
E.2	Rozdělení beta_uni	71
E.3	Rozdělení const_norm	71
E.4	Rozdělení gamma_2norm	71
E.5	Rozdělení same_triang	72
E.6	Rozdělení same_uni	72
E.7	Rozdělení triang_beta	72
E.8	Rozdělení triang_weibull	72
E.9	Násobička mul8u_R36	72

E.10 Násobička mul8u_17MN	72
E.11 Násobička mul8u_1A0M	73
E.12 Násobička mul8u_12KA	73
E.13 Zpoždění násobičky mul8u_R36	73
E.14 Zpoždění násobičky mul8u_12KA	73

Kapitola 1

Úvod

Tato bakalářská práce se zabývá statistickým ověřováním modelů přibližných násobiček. Přibližné (neboli aproximační) výpočetní systémy jsou technologií, která reaguje na sítici poptávku po energeticky úsporných zařízeních a která se v posledních letech těší významnému rozvoji. Aproximační násobičky nabízejí – za cenu zavedení určité chyby do výpočtu – zmenšení plochy obvodů oproti přesným násobičkám, čímž lze dosáhnout jak energetické, tak časové úspory. Tyto násobičky jsou vhodné pro použití v aplikacích, které jsou odolné vůči chybám.

Při verifikaci hodnotících metrik aproximačních obvodů jsou obvykle uvažovány všechny kombinace jejich vstupních hodnot. Cílem této práce je prozkoumat, jakým způsobem se mění vlastnosti vybraných aproximačních násobiček při jejich použití v kontextu různých aplikací. V rámci práce je zvoleno několik algoritmů, u nichž dochází k násobení celých kladných čísel. U těchto algoritmů je poté analyzována četnost výskytů jednotlivých dvojic násobených čísel, z čehož jsou následně odvozena pravděpodobnostní rozdělení, která jsou použita při generování vstupů zkoumaných aproximačních násobiček. Jednotlivé násobičky jsou poté simulovány s výše zmíněnými rozděleními. Při tom jsou pozorovány různé hodnotící metriky zaměřující se především na míru chybovosti daných násobiček.

Druhá kapitola se věnuje přibližným výpočetním systémům. Po uvedení jejich základního principu se zaměřuje na jednotlivé oblasti, v nichž aproximační systémy využívají uplatnění, a krátce také na oblasti, kde naopak využití aproximačních systémů vhodné není. Následuje představení některých používaných aproximačních technik. Dále jsou uvedeny hodnotící metriky pro posuzování vlastností aproximačních systémů. Závěrečná část kapitoly se zaměřuje na různé přístupy k tvorbě aproximačních násobiček.

Třetí kapitola se zabývá statistickým ověřováním modelů. Nejprve jsou vysvětleny základní pojmy z oblasti verifikace systémů. Následuje srovnání klasického a statistického ověřování modelů. Poté jsou v kapitole uvedeny klasické a stochastické časové automaty. Na závěr je představen modelovací nástroj UPPAAL.

Ve čtvrté kapitole je uveden návrh a postup řešení praktické části bakalářské práce. Nejprve jsou popsány jednotlivé používané modely. Dalé je popsána implementace převodu modelů násobiček z Verilogu do modelů v prostředí UPPAAL. Následuje výpis zvolených aplikací a od nich odvozených pravděpodobnostních rozdělení. Poté jsou uvedeny použité simulační dotazy.

Pátá kapitola prezentuje získané výsledky simulací, včetně porovnání některých výsledků s referenčními hodnotami z knihovny EvoApproxLib.

Šestá kapitola se zabývá škálovatelností zvoleného systému s ohledem na časovou a paměťovou náročnost.

Kapitola 2

Přibližné výpočetní systémy

S rostoucím využitím výpočetních technologií v mnoha různých oblastech lidské činnosti (v posledních letech zejména rapidní rozvoj strojového učení, zpracování velkých dat, aj.), rostou také nároky na výkon a efektivitu počítačů. Zároveň se zvyšují i nároky na mobilitu výpočetních systémů, mnoho zařízení obsahuje vestavěné systémy, což omezuje možnou velikost těchto systémů.

Dle Moorova zákona se počet tranzistorů na jednom čipu každé 2 roky zhruba zdvojnásobí, ten ale pravděpodobně v dohledné době přestane vzhledem k fyzikálním vlastnostem tranzistorů platit [23]. Z tohoto důvodu je nutné se zamýšlet nad efektivnějším využitím výpočetních systémů. Jedním z možných přístupů, který se v posledních letech těší velkému rozvoji jak ve výzkumu, tak v realné implementaci, je využití přibližných (aproximačních) výpočetních systémů.

V této kapitole bude nastíněn úvod do problematiky přibližných výpočetních systémů. Dále budou uvedeny oblasti, v nichž lze approximaci použít a u kterých je naopak její použití nevhodné. Následují přehledy aproximačních technik, používaných hodnotících metrik a v závěru rozbor existujících přístupů k tvorbě aproximačních násobiček.

2.1 Využitelnost a omezení přibližných výpočetních systémů

Aproximační systémy lze typicky využít v oblastech, v nichž není nezbytně nutné znát přesný výsledek nějaké operace nebo výpočtu [10]. Takových oblastí je v informatice celá řada, zejména se jedná o zpracování multimédií, strojové učení, zpracování signálů, analýzu velkých dat, webové vyhledávače, oblast rozšířené reality, počítačové vidění, aj. Využití aproximačních technik v některých zmíněných oblastech je popsáno níže.

Zpracování signálů a obrazu

V oblasti zpracování obrazu mohou přibližné výpočetní systémy zrychlit procesy jako je např. segmentace obrazu, tedy rozdělení obrazu do částí, které korespondují s konkrétními objekty v obraze (třeba pomocí approximace algoritmů pro detekci hran) [30]. Dále lze approximaci využít při zlepšování kvality obrazu, např. při zvýraznění hran nebo při odstranění šumu.



Obrázek 2.1: Příklady výstupů approximačních prahovacích algoritmů. Převzato z [13]

Analýza velkých dat

Při zpracování tzv. velkých dat (angl. *big data*) lze approximovat dotazy na data, které jsou často velmi komplexní, výsledky jejich přibližných variant jsou ovšem mnohdy dostačující. Dále lze approximovat i samotná data za účelem snížení nároků na úložný prostor. Pro nějakou třídu dotazů Q na data D musí platit, že pro approximovaná data D' vrátí dotazy Q uspokojivé výsledky. Dotazy Q bývá obvykle potřeba mírně upravit, aby vyhovovaly approximovaným datům. V obou přístupech je třeba zvážit rovnováhu mezi efektivitou dotazů a kvalitou výsledků [16].

Strojové učení

Přibližné počítání má ve strojovém učení (dále SU) obrovské využití, protože prostředí SU má v mnoha ohledech ideální podmínky k využití approximace. Mnoho úloh SU lze redukovat na problém approximace nějaké funkce, kde daná funkce není kompletně specifikována. Samotný proces trénování modelů lze přizpůsobit tak, aby byl schopen se zotavit ze všech případných nepříznivých účinků approximace [12].



Obrázek 2.2: Přehled aproximačních technik používaných ve strojovém učení. Převzato z [12]

Neaproximovatelné oblasti

Přibližné výpočetní systémy naopak není vhodné využít tam, kde jsou z různých důvodů klíčové přesné výpočty. Typickým příkladem jsou jakékoli peněžní transakce nebo algoritmy používané ve finančním inženýrství (angl. pojem *Computational Finance*), kde by jakákoli nepřesná analýza mohla vést k velkým finančním ztrátám.

Dalším příkladem jsou tzv. bezpečnostně kritické systémy, v nichž by nepřesnosti mohly vést k ohrožení lidského života (např. systémy v automobilech, letadlech, elektrárnách apod.).

Mezi nevhodné strany approximace lze dále řadit třeba kryptografii a šifrování, high-precision manufacturing (vysoce přesná výroba), klasické databázové systémy aj.

2.2 Aproximační techniky

V této sekci jsou stručně představeny některé používané aproximační techniky včetně uvedení jejich silných a slabých stránek a také konkrétních případů, kde je možné dané techniky využít [20].

Precision scaling

Škálování přesnosti je technika, při jejímž použití dochází ke snížení přesnosti aritmetických operací. Tuto techniku můžeme rozdělit na dva základní přístupy:

- Statické škálování přesnosti – úroveň přesnosti je stanovena na jednu předem danou hodnotu pro všechny výpočty. Tato úroveň bývá určena na základě očekávané tolerance spouštěné aplikace vůči šumu. Hlavní výhodou tohoto přístupu je jeho jednoduchost. Nevýhodou je možná nízká efektivita stran úspory energie nebo zlepšení výkonu oproti dynamickému škálování.

- Dynamické škálování přesnosti – oproti statickému škálování se jedná o komplexnější techniku, při níž je úroveň přesnosti výpočtů dynamicky stanovována na základě změn tolerance spouštěné aplikace vůči šumu za běhu. Snižuje tedy přesnost výpočtů v době, kdy je tolerance vůči šumu vysoká [28].

Obecně je tedy škálování přesnosti vhodné pro aplikace, které zpracovávají velké množství dat, která zároveň obsahují velké množství šumu.

Loop perforation

Perforace smyček se zaměřuje na zrychlení provádění smyček v programu pomocí odstranění některých iterací dané smyčky. Prvním krokem při používání této techniky bývá identifikace takových smyček, jejichž zkrácením či zjednodušením nedojde k výraznému zhoršení funkčnosti programu. K tomu lze využít specializovaný kompilátor, který na základě hranice zkreslení dané uživatelem tyto smyčky identifikuje a program následně upraví [1].

Tato technika má využití v oblastech numerických výpočtů, zpracování obrazu a signálu nebo v simulacích Monte Carlo.

```
for( i = 0; i < h; i+=4 ) {
    /* ... */
}

for( i = 0; i < h; i+=4 ) {
    if (doPerforate(i, environment)) continue;
    //...
}
```

Load Value Approximation

V případě, že se procesoru nepodaří načíst data z mezipaměti (anglicky tzv. *cache load miss*), je procesor nucen přistoupit k vyšším úrovním mezipaměti nebo přímo do paměti, čímž vzniká větší latence. LVA využívá aproximovatelnost některých aplikací tím, že tato data odhaduje, takže procesor může dále běžet bez větších zpoždění [20].

Příkladem využití může být technika použitelná u grafických aplikací, kdy load-value prediktor přistupuje do paměti pouze občas (narozdíl od klasických load-value prediktorů, které přistupují do paměti při každém load miss, aby ověřily správnost své predikce). Díky tomu se výrazně sníží počet přístupů do paměti, zatímco prediktor se přesto natrénuje kvalitně [18].

Memoization

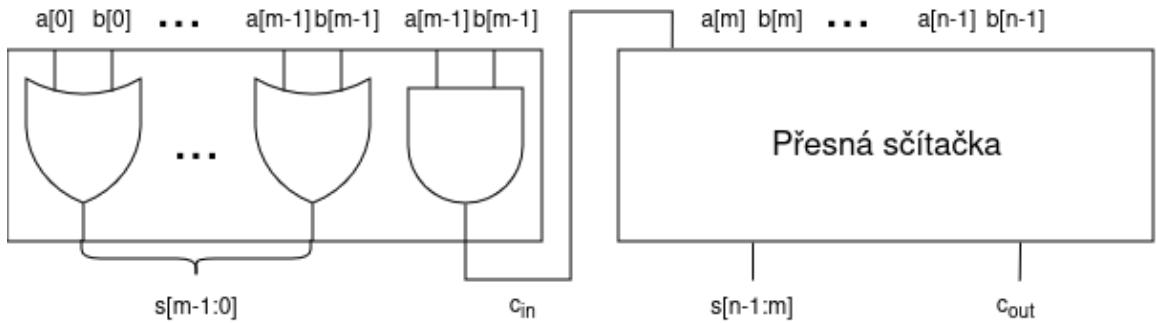
Memoizace spočívá v ukládání výstupů funkcí za účelem jejich opětovného použití při volání funkcí se stejným vstupem. Tuto metodu lze dále aproximovat znovupoužíváním těchto výstupů i u funkcí s podobným vstupem. Dochází tedy ke zrychlení díky menšímu počtu výpočtů, avšak na úkor větší paměťové náročnosti.

Memoizace je nejlépe použitelná v aplikacích, v nichž se často opakují stejné vstupy, respektive výpočty. Příkladem mohou být různé rekurzivní algoritmy, např. výpočet Fibonaciho posloupnosti, kombinatorické vzorce nebo vyhledávací algoritmy. Přibližnou variantu memoizace lze využívat v moderních grafických aplikacích, v nichž by standardní memoizace nepřinesla významné zrychlení [20].

Inexact Hardware

Využití nepřesného hardwaru (angl. *inexact hardware*) spočívá v kontrolovaném zavádění chyb do výpočetních obvodů za účelem ušetření energie, zvýšení výkonu či zmenšení plochy obvodu. Toho lze docílit např. zjednodušováním aritmetických obvodů (k tomu více v podkapitole 2.4).

Na obrázku 2.3 je znázorněn příklad přibližné sčítáčky využívající pouze OR hradla k sečtení nejméně významných bitů [8].



Obrázek 2.3: Přibližná sčítáčka lower-part-or s m nepřesnými byty. Převzato z [8]

Voltage Scaling

Voltage Scaling (škálování napětí) je aproximační technika na úrovni obvodů. Jedná se o přístup, který umožňuje dynamicky měnit napětí dodávané do elektronických komponent, jako jsou mikroprocesory nebo grafické čipy, v závislosti na aktuálních požadavcích na výkon. Zvyšování napětí, tzv. **overvolting**, se používá pro zvýšení výkonu, **undervolting**, tedy snižování napětí, je využit při snaze o úsporu energie.

Při snižování napětí v obvodech mohou vznikat chyby. Například snižování napětí u paměti SRAM může vést vedle ušetření energie k častějším neúspěšným čtením a zápisům do paměti (*read upset* a *write failure*) [20].

2.3 Hodnotící metriky

Jednou z klíčových otázek při implementaci aproximačních systémů je vyhodnocení míry chybovosti těchto systémů. **Chybové metriky** porovnávají výsledky přesných systémů s výsledky jejich aproximačních variant z hlediska dané metriky. Seznam některých často používaných metrik je vypsán níže [21] a [9]. Symbol n_i v rovnících značí počet primárních vstupů, O_{approx} a O_{orig} značí výstupy přibližných, respektive přesných systémů.

Hammingova vzdálenost (angl. Hamming distance, zkr. HD) mezi dvěma bitovými sekvencemi je rovna počtu pozic, na kterých se byty obou sekvencí liší [25]. Mezi používané varianty této metriky patří *Průměrná Hammingova vzdálenost* nebo *Maximální Hammingova vzdálenost* (v angličtině se používá výraz *bit-flip error*).

Pro posuzování chybovosti násobiček není tato metoda příliš vhodná. Pokud je například očekávaný přesný výsledek výpočtu $64_{10} = 01000000_2$ a výsledek approximace $63_{10} =$

0011111_2 , jejich Hammingova vzdálenost je 7, zatímco relativní chyba je cca. 1,5%.

$$HD = \sum_{\forall i} \text{OnesCount}(O_{approx}^{(i)} \oplus O_{orig}^{(i)}) \quad (2.1)$$

Pravděpodobnost chyby (angl. error probability, zkr. EP) značí, s jakou pravděpodobností nebude výstup approximačního systému odpovídat výstupu přesného systému.

$$EP = \frac{\sum_{\forall i: O_{approx}^{(i)} \neq O_{orig}^{(i)}} 1}{2^{n_i}} \quad (2.2)$$

Průměrná absolutní chyba (angl. mean absolute error, zkr. MAE) popisuje průměrný rozdíl mezi přesnými a přibližnými výstupy.

$$MAE = \frac{\sum_{\forall i} |O_{approx}^{(i)} - O_{orig}^{(i)}|}{2^{n_i}} \quad (2.3)$$

Průměrná kvadratická chyba (angl. mean squared error, zkr. MSE) počítá průměr druhých mocnin rozdílů mezi přesnými a přibližnými výstupy. MSE se často používá při výpočtu tzv. PSNR (zkratka z anglického *Peak signal-to-noise ratio*, česky Špičkový poměr signálu k šumu), pomocí kterého lze určovat kvalitu rekonstrukce obrázků a videí [9].

$$MSE = \frac{\sum_{\forall i} |O_{approx}^{(i)} - O_{orig}^{(i)}|^2}{2^{n_i}} \quad (2.4)$$

Průměrná relativní chyba (angl. mean relative error, zkr. MRE) uvažuje průměrnou chybu v relaci s velikostí očekávaného výstupu. Díky tomu jsou při větších hodnotách akceptovatelné větší chyby.

$$MRE = \frac{\sum_{\forall i} \frac{|O_{approx}^{(i)} - O_{orig}^{(i)}|}{\max(1, O_{orig}^{(i)})}}{2^{n_i}} \quad (2.5)$$

Nejhorší absolutní chyba (angl. worst-case error, zkr. WCE) popisuje největší možnou chybu, které je možné při approximaci dosáhnout.

$$WCE = \max_{\forall i} |O_{approx}^{(i)} - O_{orig}^{(i)}| \quad (2.6)$$

Nejhorší relativní chyba (angl. worst-case relative error, WCRE) uvažuje největší možnou chybu approximačního výstupu vzhledem k očekávanému výstupu.

$$WCRe = \max_{\forall i} \frac{|O_{approx}^{(i)} - O_{orig}^{(i)}|}{\max(1, O_{orig}^{(i)})} \quad (2.7)$$

U approximačních obvodů jsou neméně důležité jejich **fyzické vlastnosti**. Mezi základní z nich lze zařadit zpoždění, příkon a plochu obvodu. Tyto vlastnosti je možné kombinovat v různé další složené metriky, např. PDP (z angl. *Power-delay product*, tedy součin ztrát výkonu a zpoždění obvodu), ADP (z angl. *Area-delay product*, tedy součin plochy a zpoždění

obvodu) nebo EDP (z angl. *Energy-delay product*, tedy součin zpoždění a spotřebované energie obvodu) [14].

Aproximační systémy je dále možné posuzovat vhodnými **kvalitativními metrikami** vzhledem k využití daného systému v konkrétní aplikaci. Mezi často používané metriky [20] patří např. výše zmíněné **PSNR**, **SSIM** (zkratka z angl. *Structural similarity*, česky tedy *strukturální podobnost*), **Rozdíl pixelů** (angl. *Pixel difference*) nebo **UIQI** (zkratka z angl. *Universal Image Quality Index*, česky tedy *Univerzální index kvality obrazu*), které se používají při porovnávání podobnosti obrázků a videí.

Výstupy systémů, založených na strojovém učení nebo na shlukování, lze posuzovat např. na základě **Přesnosti** (angl. *Precision*), **Výtěžnosti** (angl. *Recall*), metrikou **F-score**, nebo dalších složitějších metrik [22].

2.4 Aproximační násobičky

Násobení je operace, která je při spouštění datově náročných aplikací (např. streamování, zpracování obrazu, strojové učení aj.) utilizována velmi často a která tím pádem spotřebuje nemalé množství energie. Mnohé z těchto aplikací jsou ovšem schopné vytvořit dostatečně dobrý výsledek i přes nepřesnosti v násobení. Dalším příkladem využití přibližných násobiček jsou zařízení z oblasti internetu věcí, u nichž je kladen důraz na co nejmenší spotřebu energie a u nichž také mnohdy není nutné vše počítat přesně [27].

Tato podkapitola se zaměřuje nejprve na vysvětlení funkcionality binárních násobiček a následně na popis základních přístupů k vytváření přibližných násobiček.

Základní princip binárních násobiček

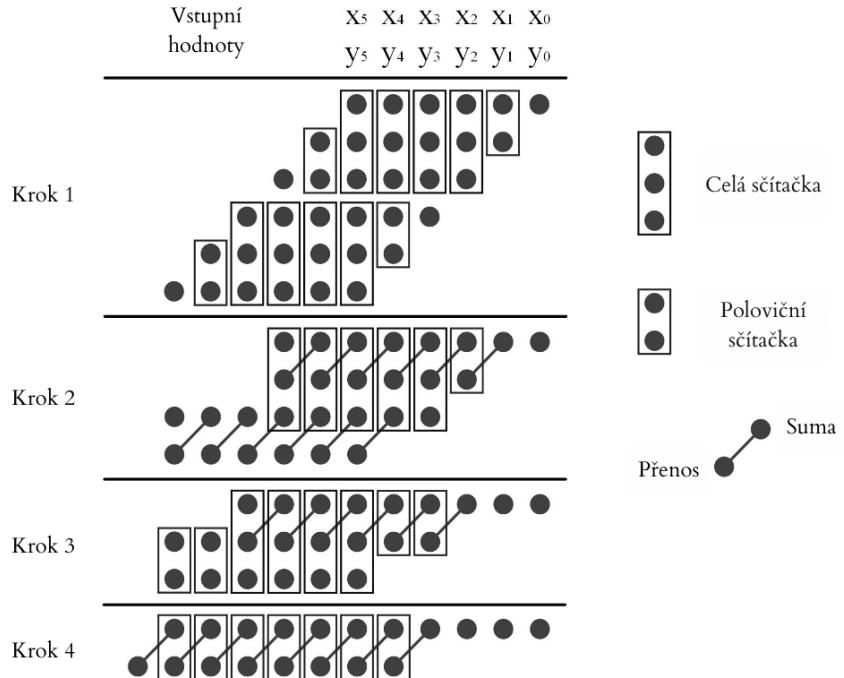
Při binárním násobení uvažujeme pouze 2 hodnoty – 1 a 0. Z toho důvodu je možné na binární násobení nahlížet jako na sekvenci sčítání a bitových posunů. Uvažme příklad z obrázku 2.4, kde vstup x je násoben vstupem y . Operaci násobení lze rozdělit do 3 fází: načtení dat na vstupu, generování částečných součinů a finální součet.

Základní binární násobička funguje tak, že se postupně roznásobují jednotlivé bity vstupu y se všemi bity vstupu x a následně na základě váhy bitu y dochází k bitovému posunu vlevo. První řádek částečných součinů v příkladu z obrázku 2.4 tedy vznikne roznásobením bitu y_0 se všemi bity vstupu x , atd. Nakonec se všechny částečné součiny sečtou v konečný výsledek.

$$\begin{array}{r}
 & \text{x}_3 & \text{x}_2 & \text{x}_1 & \text{x}_0 \\
 \times & & & & \\
 & \text{y}_3 & \text{y}_2 & \text{y}_1 & \text{y}_0 \\
 \hline
 & \text{p}_{3,0} & \text{p}_{2,0} & \text{p}_{1,0} & \text{p}_{0,0} \\
 & \text{p}_{3,1} & \text{p}_{2,1} & \text{p}_{1,1} & \text{p}_{0,1} \\
 & \text{p}_{3,2} & \text{p}_{2,2} & \text{p}_{1,2} & \text{p}_{0,2} \\
 & \text{p}_{3,3} & \text{p}_{2,3} & \text{p}_{1,3} & \text{p}_{0,3} \\
 \hline
 \text{r}_7 & \text{r}_6 & \text{r}_5 & \text{r}_4 & \text{r}_3 & \text{r}_2 & \text{r}_1 & \text{r}_0
 \end{array}
 \begin{array}{l}
 \text{vstupní} \\
 \text{hodnoty} \\
 \\
 \text{dílčí} \\
 \text{součiny} \\
 \\
 \text{výsledek}
 \end{array}$$

Obrázek 2.4: Binární násobička 4x4 bity

Násobení jednotlivých bitů je implementováno jednoduše pomocí hradel AND, kritickou sekcí procesu násobení je propagace přenosu (angl. *carry*) z nižšího bitu na vyšší bit. Přístupů k řešení tohoto problému je mnoho, jedním z nich je například tzv. ripple-carry sčítáka, kde se přenosy propagují postupně zprava doleva, nebo také tzv. carry-save sčítáka, kde se přenosy propagují diagonálně za účelem zrychlení výpočtu [27].



Obrázek 2.5: Příklad Wallace-Tree násobičky. Převzato z [27]

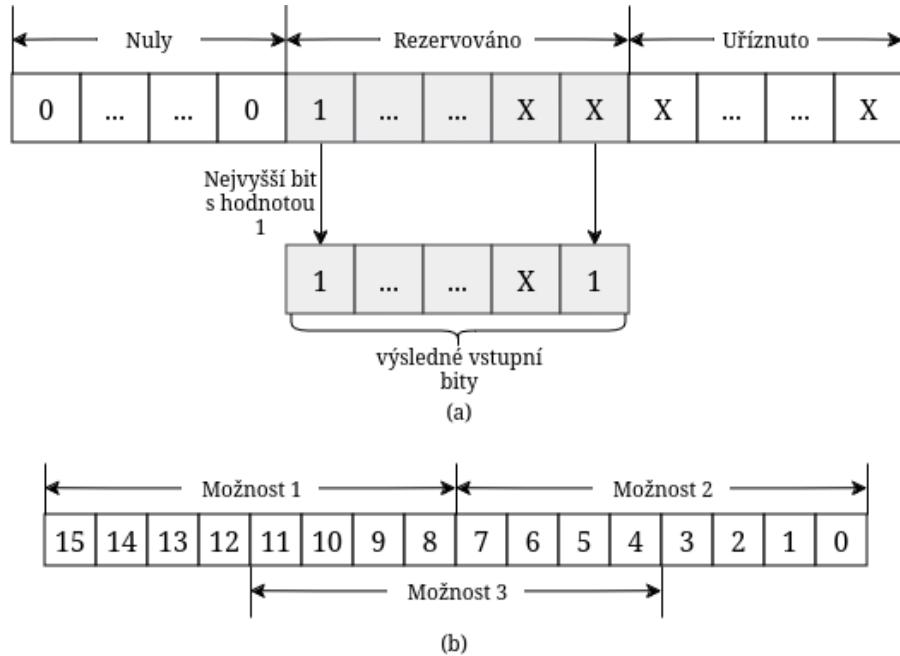
Další zrychlení přináší princip tzv. Wallace Tree, v rámci něhož se seskupují 3 částečné součiny po sloupcích, které generují 2 výstupy – sumu a přenos. Tato operace se opakuje, dokud nezůstanou poslední dva řádky částečných součinů, které se poté sečtou v konečný výsledek. V této násobičce jsou využívány úplné a částečné sčítáky, dalšího zrychlení lze docílit utilizací paralelních výpočtů [26]. Ilustrace této násobičky je na obrázku 2.5.

Aproximace vstupních hodnot

Jednoduchým a přesto efektivním způsobem approximace je approximace vstupních dat. Toho lze docílit např. uříznutím několika nejméně významných bitů (zkr. LSB z angl. *least significant bit*), což má menší vliv na celkový výsledek výpočtu, než uříznutí několika nejvíce významných bitů (zkr. MSB z angl. *most significant bit*) [27].

Existují dvě základní metody segmentace dat – dynamická a statická. Dynamická segmentace dat (DSM) spočívá v detekci prvního nenulového bitu a oříznutí následujících k bitů. Při statické segmentaci dat (SSM) dochází k volbě jedné z předem daných strategií. Na obrázku 2.6 jsou ilustrovány obě metody. U SSM jsou v tomto případě následující možnosti: k nejméně významných bitů, k nejvíce významných bitů nebo k prostředních bitů jako kompromis.

SSM zpravidla potřebuje méně hardwarových zdrojů než DSM, na druhou stranu její výsledek bývá méně efektivní, protože může obsahovat některé redundantní bity (např. nulové MSB).



Obrázek 2.6: (a) Příklad oříznutí DSM; (b) Příklad oříznutí SSM. Převzato z [27]

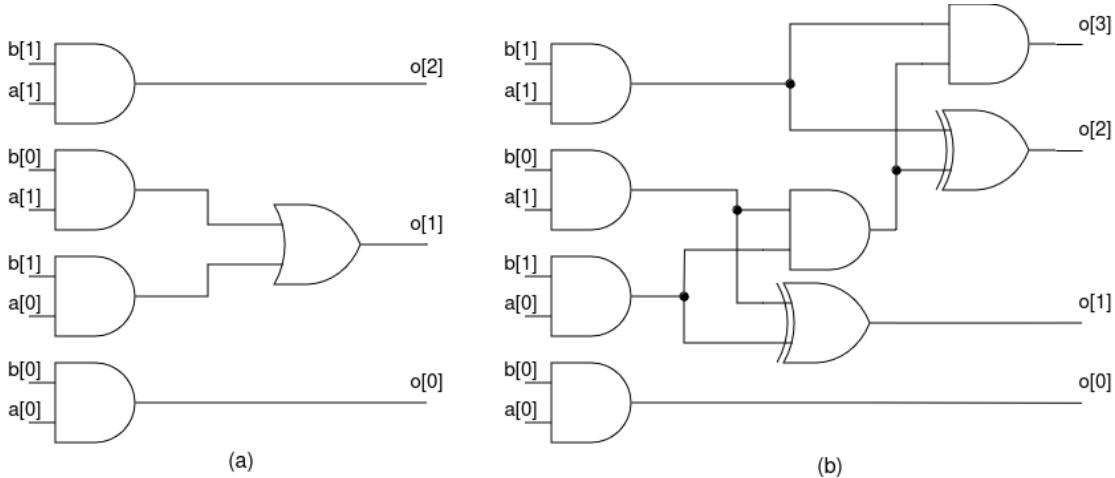
Aproximace při generování částečných součinů

Jedním z možných přístupů je využití malých bloků nepřesných násobiček k vytvoření dílčích součinů a poté sečtení těchto různě bitově posunutých dílčích součinů. Takové násobičky lze nazývat pojmem *Nedostatečně navržená násobička* neboli anglicky **Under-designed multiplier**.

Příkladem stavebního bloku může být násobička na obrázku 2.7. Jak lze pozorovat v Karnaughově mapě v tabulce 2.1, tato násobička je přesná pro 15 ze 16 možných vstupních kombinací (nepřesný výsledek je v tabulce vyznačen červeně). Změna oproti přesné násobičce je v tom, že výsledek vstupu $3 \cdot 3$ je reprezentován bity 111 oproti přesnému výsledku 1001, díky čemuž se snížila plocha obvodu téměř na polovinu (5 logických hradel oproti 9 u přesné násobičky, méně drátů) s pravděpodobností chyby pouze $\frac{1}{16}$ [15].

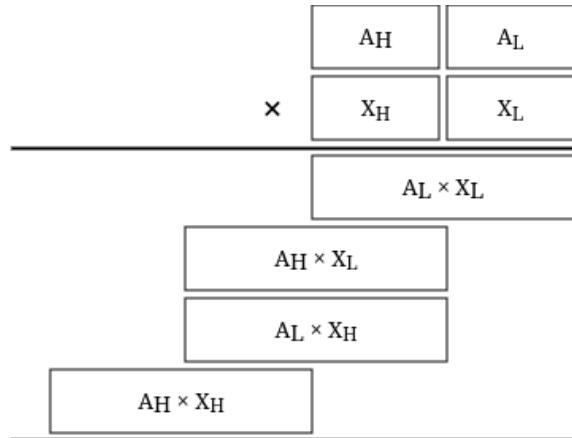
\times	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	111	110
10	000	010	110	100

Tabulka 2.1: Karnaughova mapa pro nepřesnou násobičku z obrázku 2.7



Obrázek 2.7: (a) Přibližná násobička 2x2 byty (b) Přesná násobička 2x2 byty

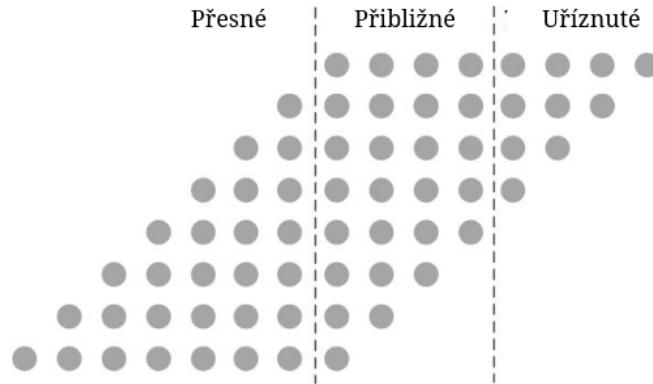
Na obrázku 2.8 je znázorněn příklad násobičky 4x4 bitů složené ze 4 bloků násobiček 2x2 bitů. A a X jsou vstupní hodnoty, dolní indexy H, respektive L značí 2 vyšší, respektive 2 nižší byty vstupu. V jednotlivých blocích postupně dochází k roznásobení všech kombinací dvojic vstupních bitů. Tyto dílčí výsledky jsou následně bitově posunuty a sečteny, čímž vzniká celkový výsledek násobení.



Obrázek 2.8: Tvorba násobiček z menších bloků. Převzato z [15]

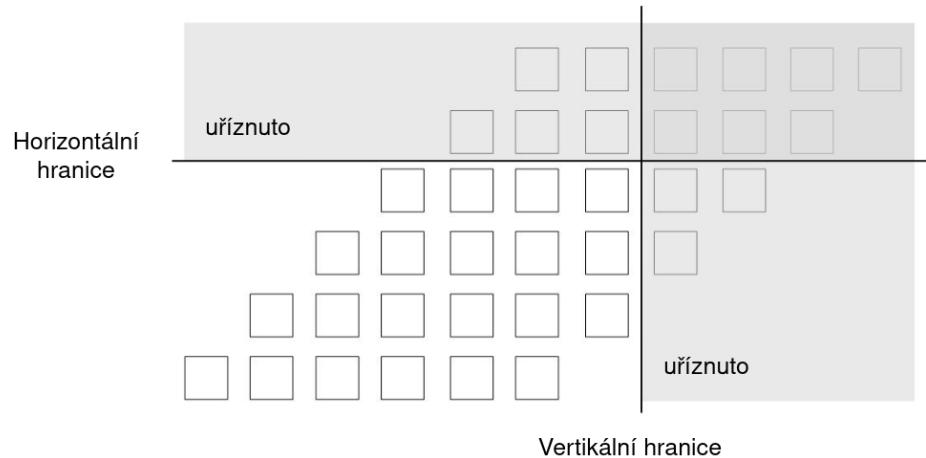
Aproximace při závěrečném sčítání

Pro závěrečné sčítání částečných součinů se používají sčítačky a kompresory, proto je přirozené uvažovat o approximaci těchto komponent za účelem approximace celé násobičky. Přístupů k tomuto řešení existuje celá řada, v posledních letech výzkumníci pracují s myšlenkou rozdělení matice částečných součinů na několik skupin (obvykle 2 nebo 3), přičemž u každé skupiny by docházelo k jinak velké approximaci. U příkladu na obrázku 2.9 by nejvíce významné byty zůstaly nezměněny, prostřední byty by podléhaly určité approximaci a nejméně významné byty by byly uříznuty nebo akumulovány pouze pomocí OR hradel.



Obrázek 2.9: Příklad rozdělení matice částečných součinů na skupiny s různou úrovní aproximace. Převzato z [27]

Další možností je rozdělení matice částečných součinů pomocí vertikálního a horizontálního uříznutí. Jak je vidět v příkladu na obrázku 2.10, všechny částečné součiny nahoru od horizontální hranice jsou při konečném součtu ignorovány. To stejné platí i pro částečné součiny napravo od vertikální hranice. Pomocí posouvání těchto hranic lze modifikovat míru approximace. Takto navržené násobičky se nazývají anglickým pojmem **Broken-Array Multiplier**, tedy násobička s rozbitou maticí částečných součinů [17].



Obrázek 2.10: Příklad násobičky typu Broken-Array. Převzato z [17]

Logaritmické násobičky

Násobičky lze approximovat také na úrovni algoritmů. Jednou z možností je tvorba logaritmických násobiček, kde lze na násobení nahlížet jako na součet logaritmů obou činitelů. Při násobení čísel $A \cdot B$ lze logaritmus pro číslo A vyjádřit následovně [19]:

$$A = 2^{k_1}(1 + x_1), \quad (2.8)$$

$$\log_2(A) = k_1 + \log_2(1 + x_1), \quad (2.9)$$

kde A je činitel, k_1 je pozice nejvýznamnějšího bitu s hodnotou 1 a x_1 je desetinná část ležící v intervalu $(0, 1)$. Stejnou formulí s parametry k_2 a x_2 lze vyjádřit i logaritmus pro činitel B . Logaritmus samotného násobení lze poté vyjádřit následujícím způsobem:

$$\log_2(A \cdot B) = k_1 + k_2 + \log_2(1 + x_1) + \log_2(1 + x_2) \quad (2.10)$$

Implementace tohoto výpočtu vyžaduje použití detektoru nejvyššího jedničkového bitu (angl. *Leading-one detector*), binárně-logaritmických převodníků (angl. *logarithm-binary converter* a *binary-logarithm converter*) a sčítáčky [27]. Pro snížení implementační náročnosti lze výpočet approximovat následovně:

$$\log_2(x + 1) \approx x, 0 \leq x < 1. \quad (2.11)$$

Tím vzniká rovnice $A \cdot B \approx 2^{k_1+k_2+x_1+x_2} = 2^{k_1+k_2} \cdot 2^{x_1+x_2}$. Na základě přenosu z výpočtu $x_1 + x_2$ může být výpočet $A \cdot B$ dále approximován jako:

$$A \cdot B \approx \begin{cases} 2^{k_1+k_2}(x_1 + x_2 + 1) & \text{pro } x_1 + x_2 < 1, \\ 2^{k_1+k_2+1}(x_1 + x_2) & \text{pro } x_1 + x_2 \geq 1. \end{cases} \quad (2.12)$$

Při porovnání s výsledkem přesného násobení (za předpokladu, že $x_1 + x_2 < 1$) lze chybu approximovaného výpočtu vyjádřit jako [27]:

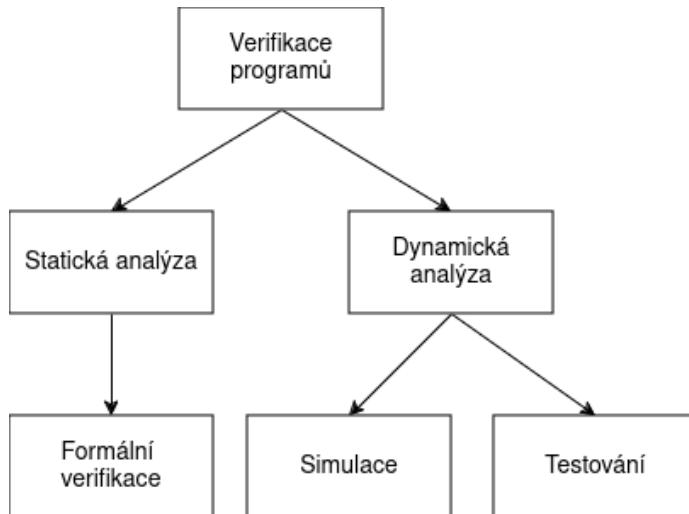
$$\begin{aligned} Chyba &= A \cdot B - 2^{k_1+k_2}(x_1 + x_2 + 1) \\ &= 2^{k_1+k_2}(1 + x_1)(1 + x_2) - 2^{k_1+k_2}(x_1 + x_2 + 1) \\ &= 2^{k_1+k_2}x_1x_2 \end{aligned} \quad (2.13)$$

Kapitola 3

Statistické ověřování modelů

Chyby v počítačových systémech mohou mít v dnešní době drastické dopady na společnost, včetně ohrožení lidských životů, proto je dokazování správnosti těchto systémů mimořádně důležitá činnost. K verifikaci počítačových systémů tradičně existují dva základní přístupy: statická a dynamická analýza (viz obrázek 3.1).

Statická analýza využívá metody formální verifikace, např. ověřování modelů, kterými lze garantovat (matematicky dokázat) bezchybnost nějakého systému. Oproti tomu dynamická analýza spočívá v simulování a testování daného systému, přičemž chyby se zachytávají při bězích jednotlivých testovacích případů. Ideální by bylo všechny systémy formálně verifikovat a zamezit tak možnosti výskytu chyby, na formální verifikaci mnohých reálných komplexních systémů ovšem nestačí výpočetní výkon současných počítačů. Tyto systémy lze ověřovat testováním, které ale pro změnu téměř nikdy neodhalí všechny chyby systému.



Obrázek 3.1: Rozdělení přístupů k verifikaci programů

Metody formální verifikace se původně zaměřovaly na diskrétní chování systémů. Výzkum v posledních letech ovšem ukazuje, že spojity (reálný) čas hraje u mnohých systémů klíčovou roli, a měl by proto být zohledněn při verifikaci. K modelování takových systémů proto vznikly časové automaty. Jedním z nejvýznamnějších nástrojů, které se prací s časovými automaty zabývají, je UPPAAL [7].

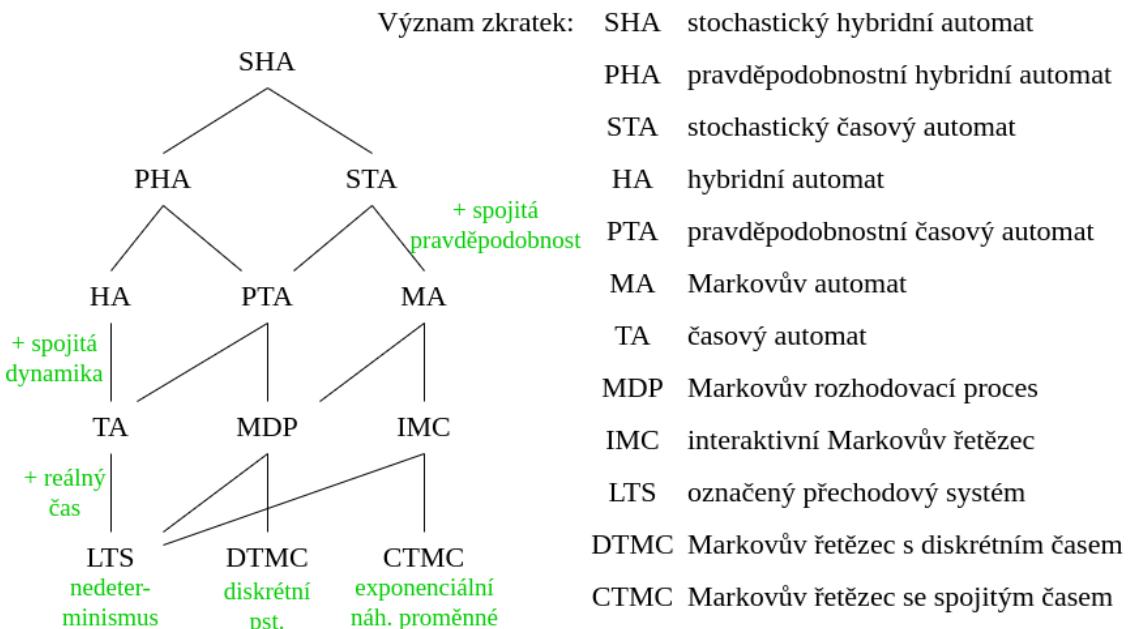
Časové automaty ovšem nejsou dostatečně expresivní na to, aby dokázaly popsat chování komplexnějších systémů s reálným časem. Chování těchto systémů často závisí na stochastických jevech, ověřování modelů takových systémů je nerozhodnutelné. S alternativním řešením tohoto problému přichází rozšíření UPPAAL SMC, které tyto systémy reprezentuje pomocí sítí stochastických časových automatů, jejichž chování může záviset na stochastických a nelineárních dynamických jevech. K efektivní analýze vlastností modelů využívá **statistického ověřování modelů**.

Zdrojem následujících 3 odstavců je primárně článek [11].

Stochastické časové automaty jsou součástí hierarchie výpočetních modelů založených na automatech, která je znázorněna ve stromě na obrázku 3.2. Modely, které jsou umístěny ve stromě výše, dělí vlastnosti od svých níže postavených předchůdců a obvykle přidávají některé funkcionality navíc.

Jak z obrázku plyne, většina těchto výpočetních modelů vychází z označených přechodových systémů, které zavádí nedeterministický výběr přechodu a také je možné pomocí nich modelovat paralelní systémy umožňující komunikaci mezi jednotlivými modely [3]. Diskrétní pravděpodobnostní rozhodování má zase původ v Markovových řetězcích s diskrétním časem. Kombinací těchto dvou vlastností vznikají Markovovy rozhodovací procesy.

Časové automaty zavádějí modelování reálného času, v kombinaci s diskrétním pravděpodobnostním rozhodováním poté vznikají pravděpodobnostní časové automaty. Stochastické časové automaty k těmto vlastnostem přidávají možnost pracovat se spojitým pravděpodobnostním rozdělením. Kombinací s hybridními automaty, které zavádějí spojitu dynamiku umožňující modelovat složité fyzikální procesy, vznikají stochastické hybridní automaty.



Obrázek 3.2: Hierarchie kvantitativních automatů, převzato z [11]

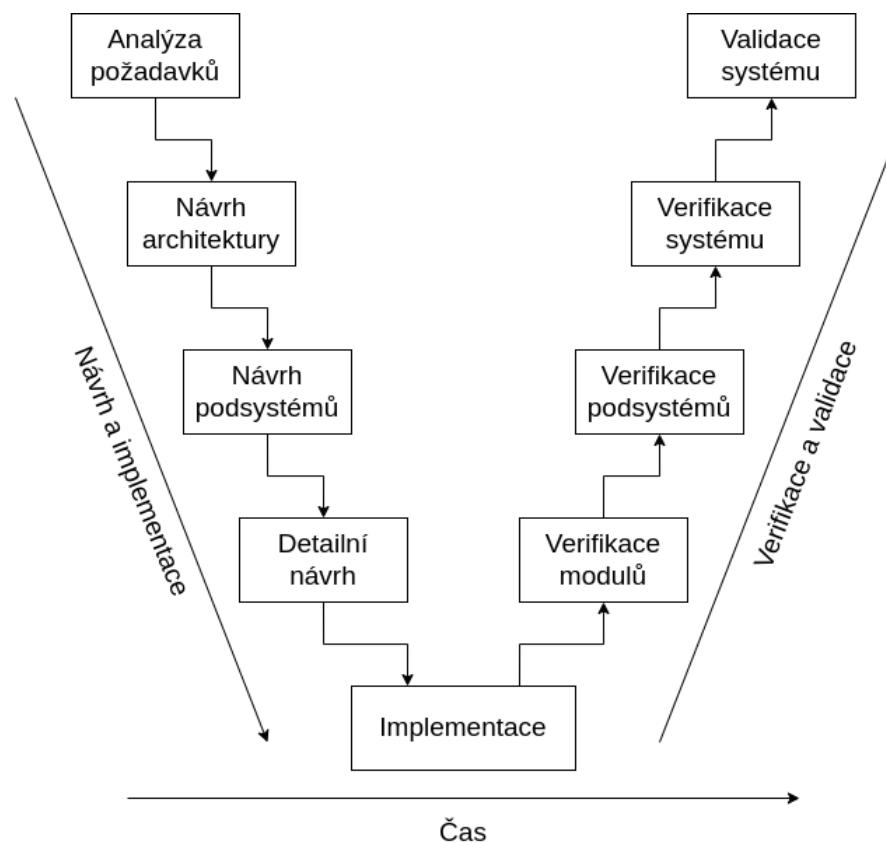
Tato kapitola se zabývá nejprve uvedením principu statistického ověřování modelů, poté definicí časových automatů a nakonec rozbořem nástroje UPPAAL včetně jeho uživatelského rozhraní, dotazovacího jazyka a dalších rozšíření.

3.1 Ověřování modelů

Tato kapitola čerpá z publikace [3].

Ověřování modelů (angl. *Model Checking*) je jeden z možných přístupů verifikace počítačových systémů. **Verifikace systému** spočívá ve snaze zajistit, že daný zkoumaný systém splňuje určité vlastnosti. Vlastnosti, které by měl daný systém splňovat, vyplývají ze **specifikace systému**. V rámci takové specifikace je definováno očekávané chování systému. Závada systému je objevena v takový okamžik, kdy není splněna některá z požadovaných vlastností systému. **Správnost systému** je vždy relativní vzhledem ke specifikaci, nejedná se o absolutní vlastnost systému jako takového.

Verifikace jako součást vývoje software je znázorněna v rámci tzv. V-modelu na obrázku 3.3. Jak je vidět, prvním krokem vývoje systému je analýza požadavků zákazníka. Následují návrhy různých vrstev systému a vznikají první prototypy. Poté dochází k verifikaci jednotlivých částí systému, ať už pomocí ověřování modelů, testování či jiných metod. Neméně důležitá je **validace systému**, při níž je ověřováno, že systém skutečně splňuje požadavky dané zákazníkem.



Obrázek 3.3: V-model popisující fáze vývoje počítačového systému

Ověřování modelů je technika, v rámci níž dochází k procházení kompletního stavového prostoru zkoumaného modelu. Díky tomu je možné s jistotou tvrdit, že model splňuje či nesplňuje danou vlastnost. Ze skutečnosti, že je procházen celý stavový prostor, vyplývá i největší nevýhoda ověřování modelů – pro složité systémy s velkým počtem stavů může být klasické ověřování modelů příliš výpočetně a paměťově náročné.

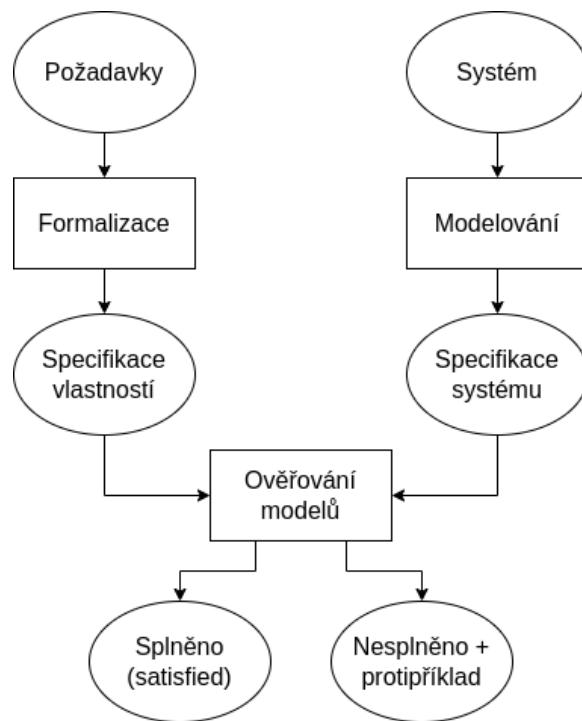
Proces ověřování modelů je ilustrován na obrázku 3.4.

Ve **fázi modelování** dochází nejprve k vytvoření modelu systému v daném modelovacím jazyce. Ke zběžnému zhodnocení vytvořeného modelu je vhodné provést několik simulací. Také je potřeba formalizovat požadavky na systém v rámci jazyka určeného pro specifikaci vlastností.

Ve **fázi běhu** dochází ke spouštění programu pro ověřování modelů nad jednotlivými dotazy za účelem ověření formalizovaných požadavků.

Ve **fázi analýzy** jsou vyhodnocovány výsledky jednotlivých ověřovacích dotazů. V případě, že zkoumaná vlastnost nebyla splněna, dojde k vygenerování tzv. **protipříkladu**. Protipříklad je vygenerovaná stopa (průchod jednotlivých stavů systému), která demonstriuje porušení zkoumané vlastnosti.

Jakmile jsou všechny vlastnosti ověřeny a splněny, lze modelovaný systém označit za správný.



Obrázek 3.4: Schéma principu ověřování modelů, převzato z [2]

3.1.1 Alternativní verifikační techniky

V této podsekci jsou stručně představeny některé další techniky verifikace počítačových systémů.

Verifikace software

- Testování – spočívá v opakovém spouštění programů (jak celých výsledných programů, tak jejich částí) s různými vstupními parametry a různými vnějšími podmínkami za účelem objevení možných chyb v systému.
- Peer Review (vzájemné hodnocení) – kontrola zdrojových kódů prováděna týmem softwarových inženýrů, který se v nejlepším případě nepodílel na vývoji daného soft-

waru. Při těchto kontrolách nedochází k překladu kódu ani ke spouštění výsledné aplikace, kód je analyzován staticky.

Verifikace hardware

- Emulace – jedná se o druh testování, při němž je nějaký rekonfigurovatelný obecný hardwarový systém (emulátor) nastaven tak, že jeho vlastnosti odpovídají vlastnostem zkoumaného systému. Poté je podrobován testům s různými vstupními parametry.
- Simulace – v případě simulací je model daného obvodu zkonstruován a poté jsou na něm prováděny simulace. Modely obvodů jsou obvykle napsány v jazycích určených pro popis hardwaru (*hardware description language*), jako jsou např. Verilog nebo VHDL.
- Strukturální analýza – kombinuje různé ověřovací techniky, jako např. syntézu (*Synthesis*), časovou analýzu (*Timing Analysis*) nebo kontrolu ekvivalence (*Equivalence checking*).

3.2 Úvod do statistického ověřování modelů

Tato sekce čerpá z článku [7].

Statistické ověřování modelů (SMC) představuje určitý kompromis mezi testováním a klasickými formálními technikami ověřování modelů. Jedná se o techniku, která spočívá v monitorování simulací nějakého systému se zaměřením na určité vlastnosti a poté využití výsledků simulací k získání statistik sloužících k odhadu správnosti daného systému. Mezi používané statistické metody se řadí sekvenční testování hypotéz (*sequential hypothesis testing*) nebo simulace Monte Carlo.

SMC nachází využití u systémů, které jsou příliš komplikované pro tradiční metody ověřování modelů z důvodu exploze stavového prostoru. Konkrétně se SMC využívá při ověřování systémů v biologii [29], systémů zaměřujících se na optimalizaci spotřeby energie [6], v leteckém a automobilovém průmyslu i jinde. Důvodem tohoto úspěchu je relativně jednoduchá implementace a snadné pochopení a používání i pro uživatele, kteří nejsou výzkumníky v oblasti simulací a ověřování modelů. Využití statistiky také umožňuje aproximovat řešení jinak nerozhodnutelných problémů.

Mezi hlavní výhody statistického ověřování modelů lze zařadit:

- Škálovatelnost – díky tomu, že SMC neprochází celý stavový prostor, je možné ověřovat i systémy s velmi velkým až nekonečným stavovým prostorem.
- Efektivnost – je výpočetně efektivnější pro systémy se složitým a stochastickým chováním, protože využívá spíše statistické odvozování oproti symbolickým výpočtům.
- Flexibilita – SMC může být aplikováno na celou řadu diskrétních, spojitéh i hybridních systémů.

Statistické ověřování modelů má přes své nesporné výhody i několik omezení. Přesnost výsledků závisí jak na počtu simulací, tak na správné formě dotazu a také využití správné statistické metody. Jevy, které se v chování systému objevují pouze vzácně, nemusejí být při nízkém počtu simulací objeveny a zohledněny při odvozování výsledků.

3.3 Časové automaty

Celá tato sekce čerpá z publikace [3].

Časové automaty modelují chování **časově kritických systémů** (angl. *time-critical system*). To jsou takové systémy, jejichž správná funkčnost nezávisí pouze na logickém výsledku nějakého výpočtu, ale také na čase, ve kterém daný výsledek vznikl. Může se jednat o ovladače zařízení v počítačích, různé komunikační protokoly a obecně systémy, které musí na něco reagovat v rámci určitého časového úseku. Díky zavedení časových omezení do automatů lze poté modelovat tvrzení jako např. „Semafor se přepne na zelenou v rámci následujících 30 sekund.“ Čas lze v rámci časových automatů uvažovat jak diskrétní, tak spojity.

Časové automaty tedy zavádějí časové proměnné neboli hodiny, které se liší od běžných proměnných tím, že lze pouze sledovat jejich hodnotu nebo je resetovat na 0 (tedy nelze je nastavit na libovolnou hodnotu). Omezení, která jsou závislá na čase (tedy na hodnotách hodin), lze nazývat časovými omezeními (v angličtině **clock constraints**). Každé takové omezení odpovídá gramatice

$$g ::= x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g,$$

kde $c \in \mathbb{N}$ a x je proměnná z množiny hodin. Časová omezení, která neobsahují konjunkce, jsou atomická.

Časová omezení lze přiřazovat jak k lokacím, tak k hranám automatů, v obou případech se jejich význam liší. Časová omezení u lokací se nazývají pojmem **invariant** a značí maximální dobu, kterou může automat v dané lokaci strávit. Časovým omezením u hran se říká strážci (angl. **guard**). Ta udávají, po jaké době je možné učinit přechod po dané hraně do další lokace (tedy jak dlouho minimálně musí automat strávit v lokaci, z níž hrana vychází).

Časový automat je uspořádaná osmice $TA = (Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L)$, kde

- Loc je konečná množina lokací,
- $Loc_0 \subseteq Loc$ je množina počátečních lokací,
- Act je konečná množina akcí,
- C je konečná množina hodin,
- $\hookrightarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ je relace přechodu,
- $Inv : Loc \rightarrow CC(C)$ je funkce přiřazení invariantu (*invariant-assignment function*),
- AP je konečná množina atomických tvrzení (*atomic proposition*),
- $L : Loc \rightarrow 2^{AP}$ je funkce označení (názvů) lokací.

Intuitivní reprezentace přechodu $\ell \xrightarrow{g:\alpha,D} \ell'$, kde g je časové omezení, α je akce a D je množina hodin, je následující: Automat přechází z lokace ℓ do lokace ℓ' pokud je splněno časové omezení g . Dále je provedena akce α a všechny hodiny z množiny D jsou resetovány na hodnotu 0. Příklad znázornění jednoduchého časového automatu je možné vidět v následující sekci na obrázku 3.7.

3.3.1 Stochastické časové automaty

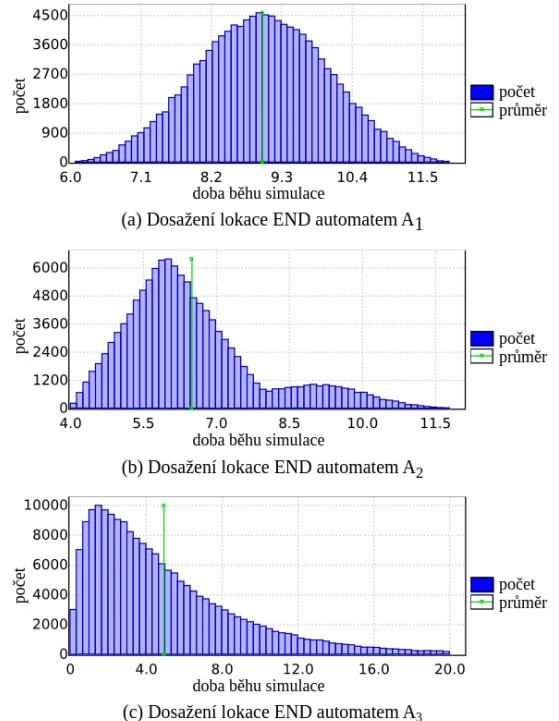
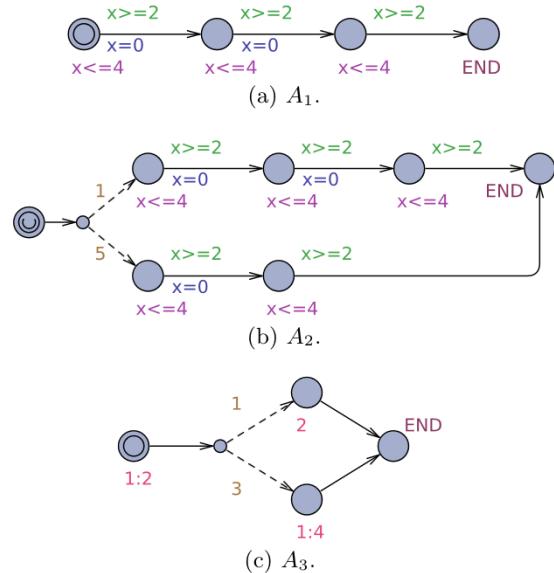
Tato podsekce čerpá z článku [7].

Stochastické časové automaty (STA z angl. *stochastic timed automata*) rozšiřují klasické časové automaty o pravděpodobnostní rozhodování. Oproti klasickým časovým automatům, u nichž je rozhodování o přechodu do další lokace nedeterministické, dochází u STA k rozhodování dle pravděpodobnostních rozdělení, která mohou či nemusí být dána uživatelem.

Podobně nedeterministické rozhodování o zpoždění času je obohaceno o pravděpodobnostní rozdělení. U ohraničených zpoždění se jedná o uniformní rozdělení, u neohraničených zpoždění poté o exponenciální rozdělení.

Na obrázku 3.5 jsou vidět 3 příklady stochastických časových automatů. Jak vyplývá z časových omezení u lokací a hran jednotlivých automatů, dosažitelnost stavu END se pohybuje v intervalech (a) $\langle 6, 12 \rangle$, (b) $\langle 4, 12 \rangle$, resp. (c) $\langle 0, \infty \rangle$. Pravděpodobnostní rozdělení dosažitelnosti lokace END pro jednotlivé automaty lze pozorovat na obrázku 3.6. U automatů A_1 a A_2 se jedná o sečtení rovnoměrných rozdělení časových omezení jejich jednotlivých přechodů. Automat A_3 neobsahuje žádná časová omezení, proto je zpoždění v jednotlivých lokacích dán exponenciálním rozdělením, v tomto případě s rychlostí danou uživatelem (hodnoty $\frac{1}{2}$, 2 , $\frac{1}{4}$).

U automatů A_2 a A_3 je také možné pozorovat větvení (tzv. *branch point*), v obou případech hned za počáteční lokaci. V těchto větvících bodech je možné definovat, s jakou pravděpodobností se provede přechod do jedné z následujících lokací. Např. u automatu A_2 se jedná o hodnoty 1 a 5, tedy do horní části se automat dostane s pravděpodobností $\frac{1}{6}$, do spodní pak s pravděpodobností $\frac{5}{6}$.



Obrázek 3.5: Příklady stochastických časových automatů, převzato z [7]

Obrázek 3.6: Rozdělení dosažitelnosti lokace END, převzato z [7]

3.4 Modelovací nástroj UPPAAL

UPPAAL je program určený k modelování, simulaci a verifikaci systémů s reálným časem. Byl vyvinut za spolupráce výzkumníků z univerzit Aalborg (Dánsko) a Uppsala (Švédsko). První vydání vzniklo v roce 1995, v době vzniku této bakalářské práce byla aktuální verze 5.0 vydaná v červnu 2023. Uživatelské rozhraní je napsáno v jazyce Java, zbytek v C++. Na C++ je také postaven jazyk používaný k deklaracím a definicím součástí modelů (k tomu více níže).

Kromě rozšíření UPPAAL SMC využívaného v této práci UPPAAL nabízí rozšíření UPPAAL Stratego zaměřující se na strategie her, UPPAAL Tiga využívající časových herních modelů (*timed game automata*) k řešení her, UPPAAL CORA (zkratka z angl. *Cost Optimal Reachability Analysis*) zabývající se efektivní analýzou dosažitelnosti modelů a UPPAAL TRON sloužící k tzv. black-box testování systémů s reálným časem.

3.4.1 Rozšíření časových automatů v nástroji UPPAAL

Modelovací jazyk programu UPPAAL rozšiřuje časové automaty o následující součásti [4]:

- Šablony (Templates) – při definici automatů je možné specifikovat parametry libovolného validního datového typu, které se automatu předají jako argumenty při inicializaci.
- Konstanty – konstantní datové proměnné, pouze integer. Deklarovány jako `const name value`.
- Ohraničené proměnné – proměnné deklarované jako `int[min, max] name`, kde `min` a `max` značí minimální, respektive maximální hodnotu proměnné. Tyto hranice jsou kontrolovány při verifikaci a překročení některé z nich vede k nevalidnímu stavu systému. Implicitní hranice jsou -32768 a 32768.
- Binární synchronizace – využívá kanály deklarované jako `chan c`. Hrana automatu označená pomocí `c!` se synchronizuje s další hrana označenou pomocí `c?`. Hrana s otazníkem čeká na signál, hrana s vykříčníkem jej vysílá. Pokud existuje více synchronizačních kombinací, je jedna z nich vybrána nedeterministicky.
- Broadcastové kanály – při broadcastové synchronizaci rozesílá jeden odesílatel `c!` signál několika příjemcům `c?` najednou. Takový kanál je deklarován jako `broadcast chan c`. Broadcastová synchronizace je neblokující, odesílatel tedy může signál vysílat i v případě, že neexistují žádní příjemci.
- Urgentní synchronizace – při urgentní synchronizaci nesmí docházet k žádnému zpoždění. Hrany, které jsou označeny kanálem urgentní synchronizace, nesmí obsahovat žádná časová omezení. Kanály urgentní synchronizace jsou deklarovány jako `urgent broadcast chan c`.
- Urgentní a zavázané lokace – popsány v podsekci 3.4.2 v části **Editor**.
- Pole – v UPPAALu je možné vytvářet pole hodin, kanálů, konstant a celočíselných proměnných.
- Funkce – uživatelské funkce lze definovat buď globálně, nebo lokálně pro jednotlivé šablony automatů. Syntax je podobná jazyku C.

3.4.2 Uživatelské rozhraní

Tato podsekce čerpá z dokumentace [5]. Uživatelské rozhraní programu se skládá ze 4 základních součástí: editor, symbolický simulátor, konkrétní simulátor a verifikátor.

Editor

Systémový editor slouží k vytváření a úpravám modelovaného systému. Popis systému je složen z množiny šablon procesů s možnými lokálními deklaracemi, dále z globálních deklarací a nakonec systémových deklarací, v nichž dochází mimo jiné k vytvoření instancí jednotlivých šablon procesů.

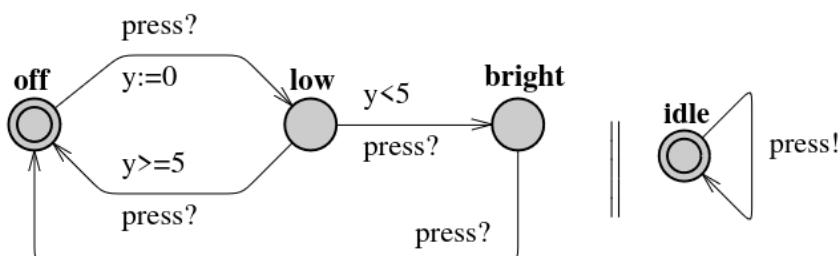
Jednotlivé procesy jsou reprezentovány pomocí časových automatů, celý systém je potom tedy síť časových automatů s možností synchronizace. Modely navíc mohou obsahovat ohraničené diskrétní proměnné, se kterými lze zacházet obdobně jako s proměnnými v programovacích jazycích (čtení, zápis, aritmetické operace). Stav systému je dán aktuální pozicí (angl. pojem *location*) jednotlivých automatů, aktuálním časem a hodnotami diskrétních proměnných.

Lokace jednotlivých automatů jsou značeny pomocí koleček, existují čtyři druhy:

- Počáteční lokace (**initial location**) – každý automat v UPPAALu má právě jednu, značena dvojitým kolečkem.
- Urgentní lokace (**urgent location**) – pokud se automat nachází v této lokaci, zastavuje se čas. Značena písmenem U uvnitř kolečka.
- Zavázaná lokace (**committed location**) – nejvíce omezující lokace, kromě zastavení času přidává podmínu, že další přechod musí být z některé ze zavázaných lokací. Značena písmenem C uvnitř kolečka.
- Normální lokace – nemá žádné speciální vlastnosti.

Na obrázku 3.7 je ilustrován jednoduchý příklad síťě automatů reprezentující ovládání lampy. Levý automat představuje lampu, pravý automat tlačítko ovládané uživatelem. Automat lampy má 3 lokace: počáteční **off**, dále **low** a **bright**. Pro synchronizaci obou automatů je využíván synchronizační kanál nazvaný **press**. Lampa čeká (**press?** u všech hran), až uživatel stiskne tlačítko (**press!**), a následně přejde do další lokace.

Lampa obsahuje také hodiny y . Pokud uživatel stiskne tlačítko několikrát za sebou dostatečně rychle ($y < 5$), může tím zvýšit jas lampy (přechod do lokace **bright**), při větší prodlevě se po druhém stisknutí lampa vypíná (vrací do lokace **off**) [4].

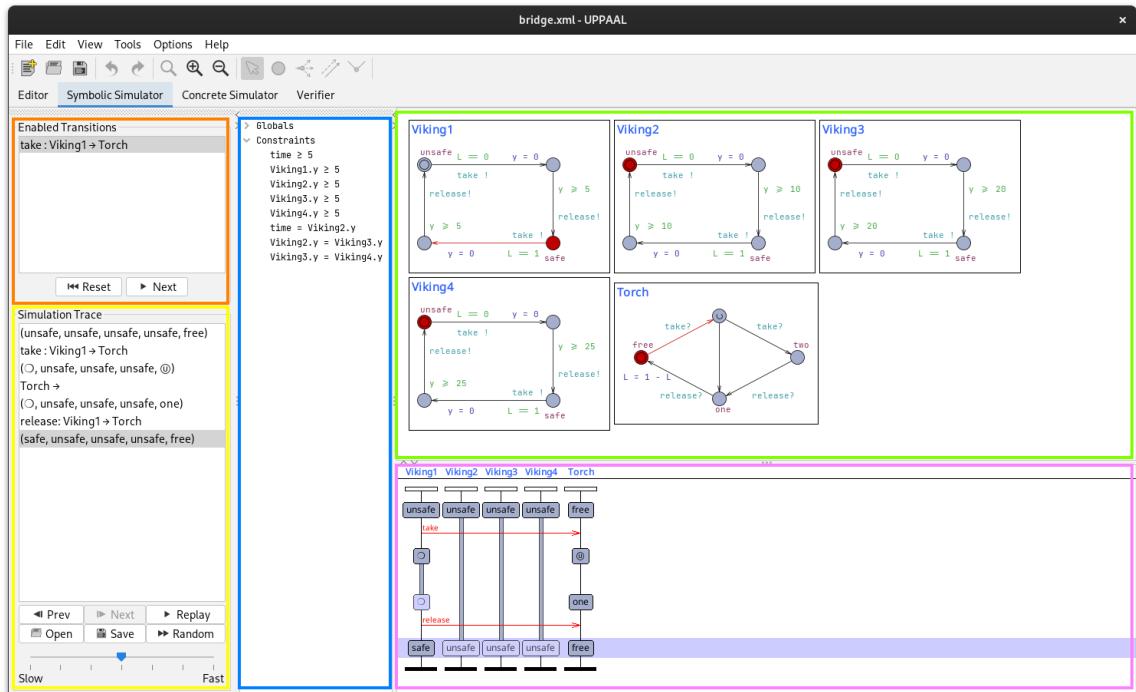


Obrázek 3.7: Jednoduchý model lampy. Převzato z [4]

Symbolický simulátor a konkrétní simulátor

Symbolický simulátor je nástroj určený k validaci modelovaného systému. Umožňuje spouštět běhy systému i na začátku návrhu (nebo při modelování) systému, čímž lze objevit chyby ještě před verifikací. Simulátor dále umožňuje vizualizovat běhy systému pomocí tzv. symbolických stop (angl. **symbolic traces**).

Časový automat se může nacházet v nekonečně mnoho různých stavech, tím pádem může vznikat i nekonečně mnoho konkrétních stop běhu systému. Simulátor není schopen všechny tyto stopy vizualizovat, proto zavádí symbolické stopy. Každý symbolický stav systému v symbolické stopě je sada stavů a jejich následovníků popsaná časovými omezeními. Aktivní lokace a hodnoty diskrétních proměnných jsou stejné pro všechny stavy v symbolickém stavu.



Obrázek 3.8: Symbolický simulátor v programu UPPAAL

Na obrázku 3.8 jsou barevně rozlišeny jednotlivé součásti simulátoru. Červená část s názvem *Enabled Transitions* slouží ke krokování simulace, uživatel si sám může vybrat, který další přechod se provede. Žlutá část *Simulation Trace* zobrazuje dosavadní vygenerovanou stopu. Dohromady tyto dva elementy slouží k ovládání simulace.

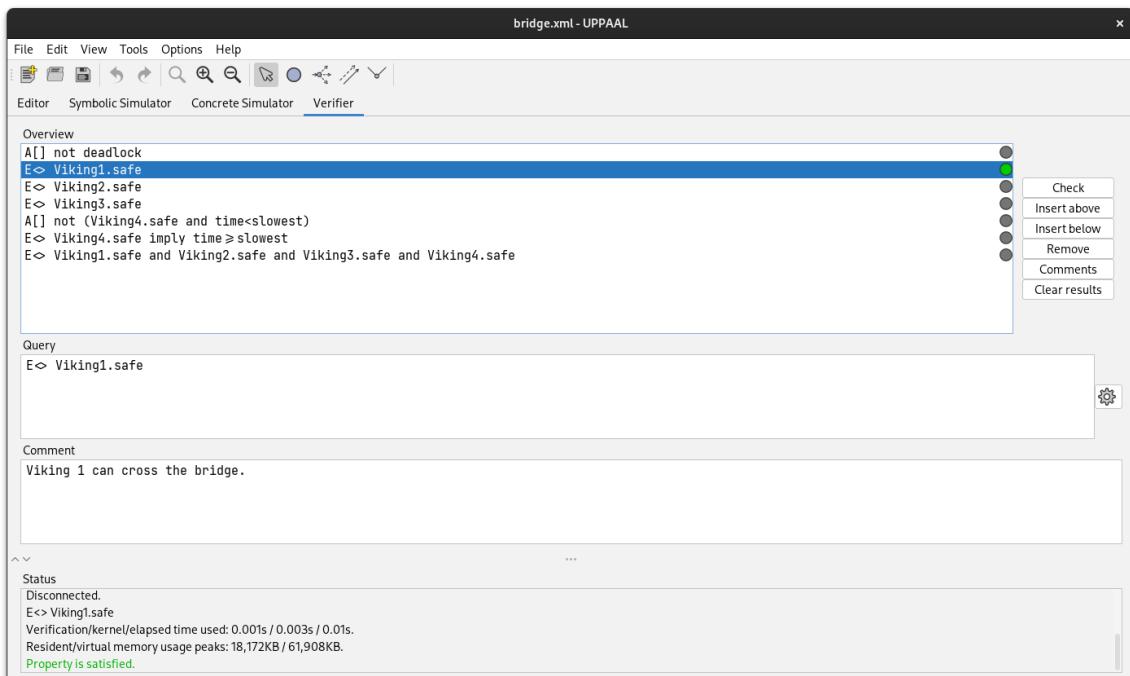
Prostřední modrý panel slouží k zobrazení hodnot globálních a časových proměnných ve stavu systému zvoleném ve výše popsané části *Simulation Trace*. Hodnoty hodin jsou zobrazeny symbolicky jako konjunkce jednotlivých časových omezení (tedy nejsou to přesné hodnoty, ale intervaly, viz např. hodnota `time` ≥ 5 na obrázku 3.8).

V zelené části jsou zobrazeny jednotlivé automaty systému. Jejich aktuální lokace ve vybraném stavu jsou zvýrazněny červenou barvou. V růžové části se nachází tzv. *Message sequence chart*, tedy graf sekvencí zpráv, v rámci něhož lze sledovat komunikaci a synchronizaci jednotlivých automatů systému.

Konkrétní simulátor funguje na podobném principu jako symbolický simulátor, tzn. také slouží k ranné validaci modelů. Odlišuje se tím, že simulace je založena na konkrétních stopách průchodů, takže uživatel může určit, v jaký přesný čas dochází k přechodu do dalšího stavu.

Verifikátor

Verifikátor slouží k ověřování bezpečnostních a živostních vlastností (*safety and liveness properties*) systému pomocí procházení symbolického stavového prostoru reprezentovaného omezeními. Dále umožňuje specifikovat a dokumentovat požadavky na systém. Na obrázku 3.9 je vidět příklad verifikátoru obsahujícího několik dotazů. Syntaxí a sémantikou dotazů se zabývají podsekce 3.4.4 a 3.4.5.



Obrázek 3.9: Verifikátor programu UPPAAL

3.4.3 Možnosti nastavení parametrů systému

Tato podsekce se zabývá možnostmi nastavení vstupních parametrů systému, jako jsou různé přístupy k procházení stavového prostoru, stanovení statistických parametrů aj. Je čerpáno z dokumentace UPPAAL [5].

Způsob průchodu stavovým prostorem (Search Order)

K procházení stavového prostoru lze zvolit jeden z následujících přístupů:

- průchod do šířky (breadth first search) – nejfektivnější varianta, pokud je třeba prohledat celý stavový prostor,
- průchod do hloubky (depth first search) – tato varianta je vhodná v případě, že uživatel očekává existenci protipříkladu,

- náhodný průchod do hloubky (random depth first search) – opět vhodné při očekávání existence protipříkladu, nalezená stopa se může kvůli randomizaci při různých bězích lišit.

Redukce stavového prostoru (State Space Reduction)

Nastavuje, do jaké míry bude program UPPAAL ukládat všechny stavy do paměti. Dochází zde ke hledání kompromisu mezi časovou a paměťovou náročností běhu. Možnosti jsou následující:

- Žádná (None) – ukládá všechny stavy do paměti,
- Konzervativní (Conservative) – neukládá tzv. zavázané stavy (*committed states*),
- Agresivní (Aggressive) – neukládá více než jeden stav za cyklus.

Reprezentace stavového prostoru (State Space Representation)

Určuje, jakým způsobem má být stavový prostor reprezentován. U některých approximativních reprezentací může docházet k tomu, že výsledek dotazu je „možná“ splněn, tedy že program UPPAAL nedokáže jednoznačně určit výsledek. Možnosti reprezentace jsou následující:

- Rozdílové matice (Difference Bound Matrices, DBM) – rychlé, u modelů s mnoha časovými proměnnými spotřebují velké množství paměti,
- Kompaktní datová struktura (Compact Data Structure) – kompaktnější (menší spotřeba paměti) a pomalejší než DBM,
- Nedostatečná approximace (Under Approximation) – využívá hashovací tabulky, míru approximace lze nastavit určením velikosti dané tabulky (možnost *Hash Table Size* v nastavení),
- Nadměrná approximace (Over Approximation) – využívá komplexní obaly k approximaci zón, nemá žádný efekt u modelů bez časových proměnných.

Statistické parametry

- **Dolní pravděpodobnostní odchylka** (Lower probabilistic deviation) ($-\delta$) – používána při testování hypotéz pro určení spodní hranice indiferenční oblasti od zadané pravděpodobnosti.
- **Horní pravděpodobnostní odchylka** (Upper probabilistic deviation) ($+\delta$) – používána při testování hypotéz pro určení horní hranice indiferenční oblasti od zadané pravděpodobnosti.
- **Pravděpodobnost falešně pozitivních výsledků** (Probability of false positives) (α) – používána při testování hypotéz a odhadu pravděpodobnosti k určení míry statistické významnosti. Jedná se o pravděpodobnost tzv. chyby typu I (*Type I error*), což je chybné zamítnutí nulové hypotézy.

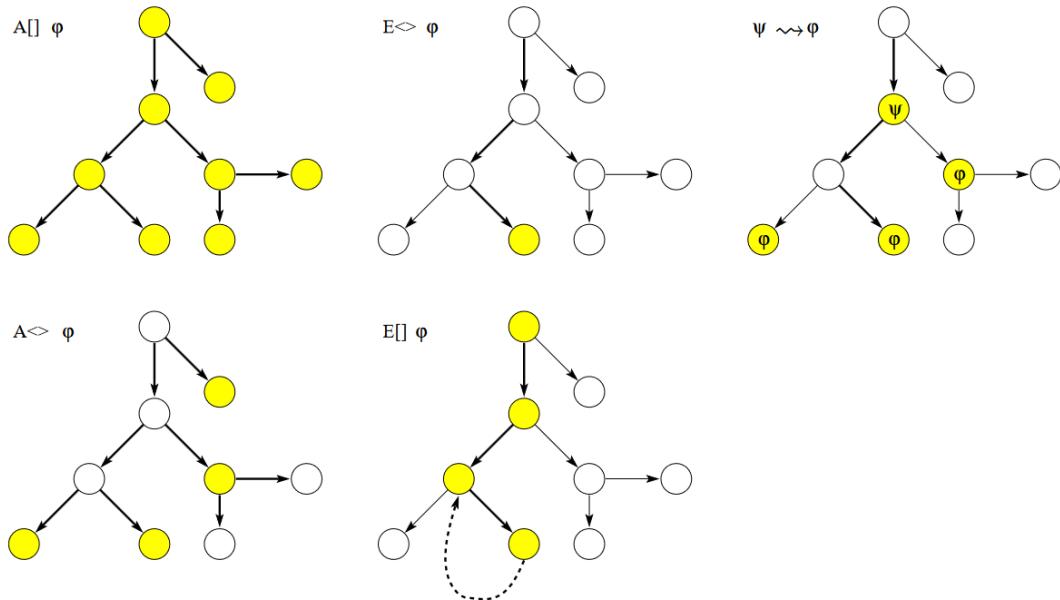
- **Pravděpodobnost falešně negativních výsledků** (Probability of false negatives) (β) – používána při testování hypotéz k určení míry statistické významnosti. Jedná se o pravděpodobnost tzv. chyby typu II (*Type II Error*), tedy chybné potvrzení nulové hypotézy.
- **Neurčitost pravděpodobnosti** (Probability uncertainty) (ε) – používána při odhadu pravděpodobnosti k omezení velikosti konfidenčního intervalu ve tvaru $p \pm \varepsilon$.
- **Dolní a horní hranice poměru** (Ratio lower bound a Ratio upper bound) (u_0, u_1)
 - používány při porovnávání dvou pravděpodobností.
- Nastavení šířky a počtu sloupců v histogramech.

3.4.4 Dotazovací jazyk

Tato podsekce čerpá z článků [4] a [7].

Hlavním úkolem nástroje na ověřování modelů je verifikace modelu s ohledem na určité požadavky. Takové požadavky musejí být vyjádřeny v nějakém formálně definovaném a strojově čitelném jazyce. Existuje několik logik, které toto splňují, UPPAAL používá zjednodušenou verzi logiky TCTL (*Timed Computation Tree Logic*). Dotazovací jazyk se tedy, podobně jako v TCTL, skládá ze stavových formulí (**state formulae**) a běhových formulí (**path formulae**). Stavové formule popisují jednotlivé stavy systému, běhové formule kvantifikují přes jednotlivé běhy nebo stopy systému.

Příklady běhových formulí lze pozorovat na obrázku 3.10. Stavy, které splňují danou stavovou formuli φ , jsou vybarveny žlutě. Hrany, které jsou procházeny při vyhodnocování, jsou ztučněny. Významy jednotlivých běhových formulí jsou vysvětleny níže ve zbytku podsekce.



Obrázek 3.10: Běhové formule v nástroji UPPAAL. Převzato z [4]

Stavové formule

Stavová formule je výraz, který může být vyhodnocen pro nějaký stav modelu bez ohledu na jeho chování. Příkladem může být jednoduchý výraz $i == 7$, který bude platit tehdy, pokud se v daném stavu i rovná 7. Syntax stavových formulí je nadmnožinou syntaxe časových omezení, u stavových formulí je navíc možné použít disjunkce. Dále je možno otěstovat, zda se proces nachází v určité lokaci pomocí výrazu $P.1$, kde P je proces a 1 je lokace.

Dosažitelnost

Dotazy na dosažitelnost se snaží zjistit, zda je daná stavová formule φ splnitelná nějakým dosažitelným stavem. Jinými slovy zda existuje cesta z počátečního stavu taková, že je během ní φ eventuelně splněno. Například při návrhu modelu nějakého komunikačního protokolu, v němž figurují odesílatel a příjemce, lze dotaz na dosažitelnost využít k ověření, že odesílatel může vůbec někdy poslat zprávu, nebo že ji příjemce eventuelně může přijmout.

K vyjádření, že by nějaký stav φ měl být eventuelně dosažitelný, lze použít běhovou formulí $E \diamond \varphi$. V syntaxi programu UPPAAL je to $E<> \varphi$.

Bezpečnost

Dotazy na bezpečnost zajišťují, že nikdy nenastane „něco špatného“. Příkladem může být sledování teploty v nějakém modelu elektrárny a nastavení hodnoty, přes kterou by se teplota nikdy neměla dostat. Jinou variantou tohoto dotazu je dotaz „Něco se možná nikdy nestane.“ Například při hraní hry je stav bezpečný, pokud v něm stále má hráč šanci vyhrát, tedy možná neprohraje.

V nástroji UPPAAL jsou tyto dotazy formulovány kladně, tedy „Něco správného vždy platí.“ Pro stavovou formuli φ lze dosažitelnost ve všech stavech vyjádřit běhovou formulí $A \Box \varphi$ (v syntaxi UPPAAL je to $A[] \varphi$). Pomocí běhové formule $E \Box \varphi$ (v syntaxi UPPAAL $E[] \varphi$) lze vyjádřit, že by měla existovat maximální cesta taková, že φ vždy platí. Maximální cesta je taková cesta, která je buď nekonečná, nebo končí ve stavu, z něhož nevedou žádné další přechody.

Živost

Dotazy na živost (angl. *liveness*) ověřují, že se něco eventuelně stane. Například pokud došlo ke stisknutí tlačítka pro spuštění televize, že se televize eventuelně spustí, nebo že zpráva, která byla odeslána v rámci nějakého komunikačního protokolu, bude eventuelně doručena.

V nejjednodušší formě lze tyto dotazy zapisovat jako $A \diamond \varphi$, tedy že stavová formule φ je eventuelně splněna. Složitější a užitečnější je forma dotazu $\varphi \rightsquigarrow \psi$ vyjadřující situaci „Vždy, když je splněno φ , bude eventuelně později splněno i ψ “. V programu UPPAAL tyto dotazy zapisujeme jako $A<> \varphi$, respektive $\varphi \dashrightarrow \psi$.

3.4.5 Rozšíření dotazovacího jazyka v rámci UPPAAL SMC

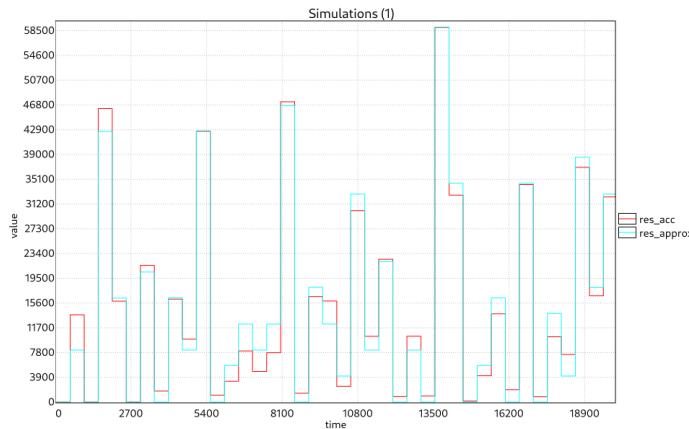
Zdrojem této podsekce je článek [7]. UPPAAL SMC rozšiřuje dotazovací jazyk o nové dotazy zaměřené na stochastickou interpretaci časových automatů. Také umožňuje uživatelům vizualizovat hodnoty výrazů a proměnných přímo v průběhu simulačních běhů. Syntax takových dotazů je následující:

```
simulate [<=bound;N] { E1, ..., Ek },
```

kde N je přirozené číslo určující počet simulací, které se mají provést, **bound** je časová hranice jednotlivých simulací a E_1, \dots, E_k jsou jednotlivé výrazy, které mají být monitorovány a vizualizovány. Příkladem může být dotaz používaný v praktické části práce k vizualizaci výsledků výpočtů přesné násobičky (proměnná `res_acc`) a zkoumané přibližné násobičky (proměnná `res_approx`):

```
simulate [<=20000;1] { res_acc, res_approx }
```

Výslednou vizualizaci je možno vidět na obrázku 3.11. Nástroj UPPAAL také umožnuje export dat ve formátu csv, čehož lze využít při další podrobnější analýze.



Obrázek 3.11: Vizualizace hodnot při simulaci, nástroj UPPAAL

Odhad pravděpodobnosti

Dotazy kvantitativního ověřování modelů mají následující syntax:

```
Pr [<=bound;N] (<>| [] expression)
```

Výsledkem takového dotazu je interval spolehlivosti odhadu pravděpodobnosti, s jakou bude běhová formule `<>| [] expression` platná za předpokladu, že platí výraz `bound`. Jako hranice `bound` lze typicky zvolit nějaké časové omezení, ať už omezení globálního času, konkrétních hodin nebo třeba počet kroků (diskrétních přechodů). Konfidenční hladinu intervalu spolehlivosti lze modifikovat parametry α a ϵ v nastavení nástroje UPPAAL.

Testování hypotéz

Testování hypotéz, neboli kvalitativní ověřování modelů, zjišťuje, zda je pravděpodobnost platnosti nějakého výrazu vyšší nebo nižší než nějaká daná hodnota (`prob_number`). Je efektivnější než dotaz na odhad pravděpodobnosti, protože je jednostranné, tudíž je potřeba méně simulací k dosažení stejně významného výsledku. Syntax je následující:

```
Pr [<=bound;N] (<>| [] expression) <=|= prob_number
```

Porovnání pravděpodobností

Tento dotaz nepřímo porovnává dvě pravděpodobnosti, aniž by je odhadoval. Syntax je následující:

$$\Pr[<=bound1; N1] \ (\><\>| [] \ expression1) \ >= \ \Pr[<=bound2; N2] \ (\><\>| [] \ expression2)$$

Odhad hodnoty

Umožňuje odhadnout maximální nebo minimální hodnotu určité proměnné (pouze integer nebo hodiny) v rámci časového omezení `bound`. Syntax je následující:

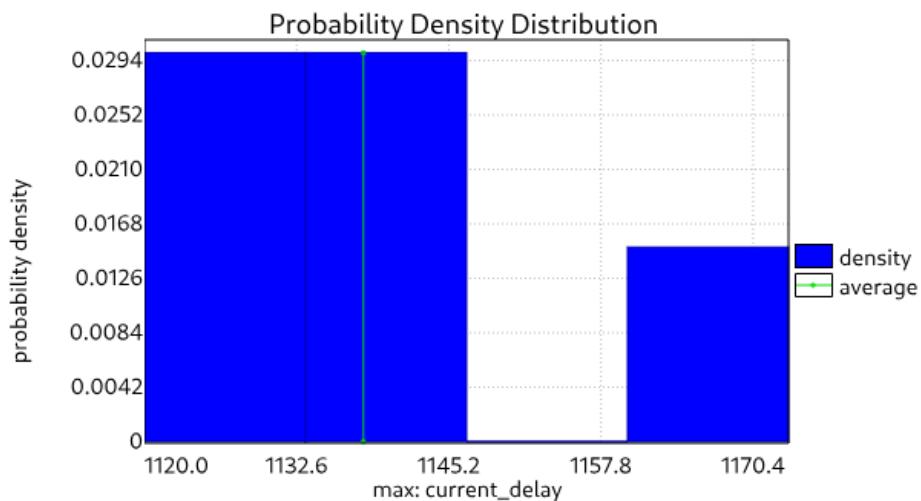
$$E[<=bound; N] \ (\min|\max: \ expression)$$

Příklad využití z praktické části práce odhaduje maximální zpoždění vzniklé při výpočtu jednoho výsledku přibližné násobičky. Časové omezení je stanoveno na 20000 jednotek, také je v dotazu explicitně stanoveno 5 simulačních běhů:

$$E[<=20000; 5] \ (\max: \ current_delay)$$

Výsledkem dotazu je konfidenční interval s předem danou hladinou spolehlivosti. Výsledkem tohoto konkrétního dotazu při simulaci jedné konkrétní náhodně vybrané přibližné násobičky byla hodnota 1138 ± 18.4169 (95% CI).

Nástroj UPPAAL dále umožňuje některé detailnější pohledy na odhad dané hodnoty. Konkrétně se jedná o graf rozdělení hustoty pravděpodobnosti (*Probability Density Distribution*), příklad takového grafu je vidět na obrázku 3.12. Dále grafy rozdělení pravděpodobnosti (*Probability Distribution*), rozdělení distribuční funkce pravděpodobnosti (*Cumulative Probability Distribution*), zobrazení konfidenčních intervalů distribuční funkce (*Cumulative Probability Confidence Intervals*) a také histogram frekvencí výskytu jednotlivých hodnot při simulačních bězích (*Frequency Histogram*). Příklady těchto grafů lze pozorovat v příloze C.



Obrázek 3.12: Grafy pravděpodobnostního rozdělení při odhadu maximální hodnoty proměnné v nástroji UPPAAL

Kapitola 4

Návrh a implementace řešení

Hlavním cílem praktické části bakalářské práce bylo vytvoření modelů zástupců zvolené třídy přibližných výpočetních systémů (angl. *Approximate Computing*, zkratka AC), ověření jejich vlastností pomocí prostředků SMC a porovnání těchto vlastností s vlastnostmi přesných variant daných výpočetních systémů.

Jako třída zkoumaných AC systémů byly zvoleny násobičky. Práce navazuje zejména na článek [24], tedy na práci vedoucího této bakalářské práce dr. Strnadela. Konkrétně přebírá systém modelů sloužící k simulaci násobení přesné a přibližné násobičky 2x2 bity a rozšiřuje jej o následující prvky:

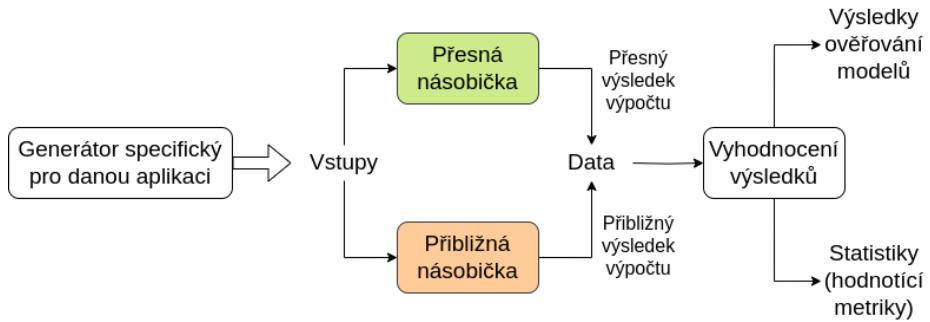
- rozšíření systému pro větší násobičky (až 11x11 bitů),
- výpočet chybových metrik popsaných v části 2.3,
- sledování počtu překlopení bitů (*bit flips*) při jednotlivých výpočtech,
- sledování zpoždění při jednotlivých výpočtech,
- generování vstupů s různým pravděpodobnostním rozdělením.

Jako zdroj modelů přibližných násobiček byla zvolena knihovna EvoApproxLib [21]. Tato knihovna poskytuje dohromady několik stovek modelů přibližných násobiček a sčítáček jak v bezznaménkové, tak znaménkové variantě. Tato práce se zaměřuje na bezznaménkové násobičky, zejména na variantu 8x8 bitů, ale implementovaný překladač (viz sekce 4.2) lze použít i pro násobičky 7x7 a 11x11 bitů.

Předmětem zkoumání bylo porovnávání vlastností přibližných násobiček při generování vstupních hodnot podle různých pravděpodobnostních rozdělení. Základní osnova postupu řešení vypadala následovně:

- nasbírání dat (násobených dvojic čísel) z vybraných algoritmů a vizualizace dat,
- tvorba pravděpodobnostních rozdělení pro generování náhodných čísel podle reálných rozdělení násobených čísel v implementacích algoritmů,
- implementace „překladače“, který převádí modely přibližných násobiček z knihovny EvoApproxLib [21] napsaných v jazyce Verilog do modelů v nástroji UPPAAL,
- vytvoření simulačních dotazů ve verifikátoru nástroje UPPAAL,

- simulace různých kombinací násobiček a náhodných rozdělení,
- zpracování a vizualizace výsledků.



Obrázek 4.1: Schéma systému porovnávání výstupů přesné a přibližné násobičky

Tato kapitola se zabývá podrobnějším popisem návrhu a řešení výše zmíněných bodů.

4.1 Popis systému v nástroji UPPAAL

Tato sekce se věnuje popisu systému modelů v nástroji UPPAAL. Zaměřuje se především na popis jednotlivých modelů – časových automatů, dále jsou zmíněny některé důležité proměnné a funkce a také je nastíněna celková synchronizace systému.

4.1.1 Globální deklarace

V rámci globálních deklarací dochází k vytvoření základního rozhraní systému. Nejdůležitější součásti jsou zobrazeny ve výpisu 4.1.

Jedná se o definici počtu vstupních a výstupních bitů systému (tedy kolik bitů mají 2 vstupní hodnoty a kolik bitů má 1 výstupní hodnota) v proměnných NPI, resp. NPO. Dále je deklarováno pole pravdivostních hodnot **bits**, které slouží jak k uložení bitů vstupních a výstupních hodnot, tak také k uložení dílčích výstupů jednotlivých logických hradel. Každé hradlo má svoje unikátní ID, které slouží jako index do pole **bits**, kam ukládá výsledek svéjí operace, a odkud jej poté mohou další hradla načíst.

Indexy bitů vstupních a výstupních hodnot jsou definovány v polích PIxy, respektive P0x (výstup přesné násobičky) a P0y (výstup přibližné násobičky).

```

1 const NPI = 16; //number of input bits
2 const NPO = 16; //number of output bits
3 const int MAX_BITS = 1024;
4 bool bits[MAX_BITS];
5
6 const int PIxy[NPI] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
7     9, 10, 11, 12, 13, 14, 15}; //indexes of bits of both input numbers
8
9 const int P0x[NPO] = {16, 17, 18, 19, 20, 21, 22,
10    23, 24, 25, 26, 27, 28, 29, 30, 31}; //indexes of bits of acc. output
11
12 const int P0y[NPO] = {32, 33, 34, 35, 36, 37, 38,
13    39, 40, 41, 42, 43, 44, 45, 46, 47}; //indexes of bits of approx. output

```

Výpis 4.1: Základní rozhraní systému

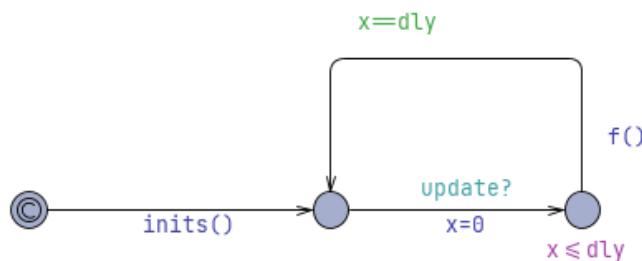
Dále zde dochází k deklaracím a definicím řady pomocných proměnných a funkcí, zejména všech proměnných obsahujících výsledky výpočtů sledovaných chybových metrik, dále také funkce porovnávající výsledky výpočtu přesné a přibližné násobičky aj.

4.1.2 Modely

V této podsekci jsou zobrazeny a popsány jednotlivé používané modely.

Model přesné násobičky

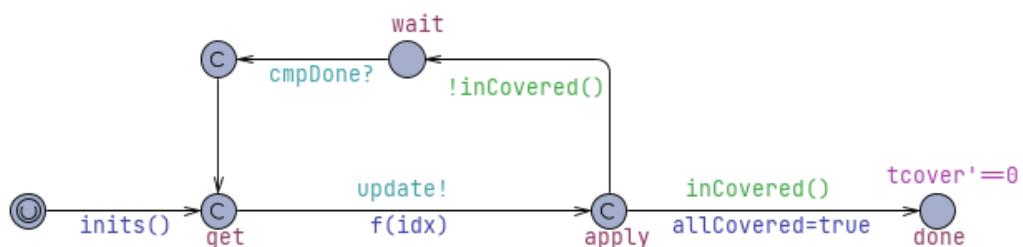
Modeluje kombinační obvod přesné násobičky definovaný pomocí pravdivostní tabulky. Po inicializaci model čeká na signál kanálu `update`, který značí vygenerování nových vstupních hodnot. Poté v rámci funkce `f()` vygeneruje výsledek výpočtu pro kombinaci vstupů a pravdivostní tabulky a uloží jej do globální proměnné `bits`. Po uběhnutí dly časových jednotek se vrací do stavu čekání na další vstupní hodnoty.



Obrázek 4.2: Časový automat reprezentující model přesné násobičky

Model generátoru vstupů

Model generátoru pseudonáhodných čísel – dvou vstupních hodnot pro přesnou i přibližnou násobičku. Po inicializaci model pomocí funkce `f(idx)` vygeneruje nové hodnoty a rozešle synchronizační signál v kanále `update`. Pomocí funkce `inCovered` sleduje, zda byly vygenerovány všechny možné vstupní kombinace. Pokud ano, ukončuje generování a přechází do lokace `done`. Jinak čeká na signál z kanálu `cmpDone` a poté generuje další dvojici hodnot.

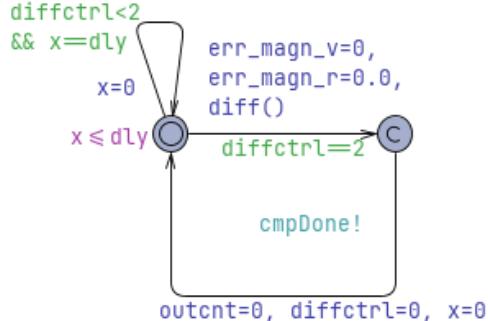


Obrázek 4.3: Časový automat reprezentující model generátoru pseudonáhodných veličin

Model kontroly rovnosti

Tento model provádí kontrolu rovnosti výsledků přibližné a přesné násobičky. Model čeká ve své počáteční lokaci, dokud nejsou připraveny výsledky jak přesného, tak přibližného

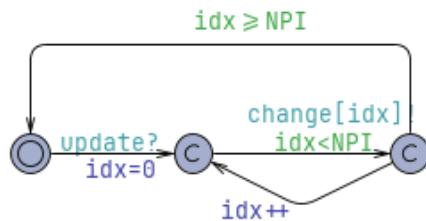
výpočtu (`diffctrl==2`). Poté v rámci volání funkce `diff()` provede porovnání výsledků a také průběžné výpočty všech sledovaných metrik. Po dokončení všech výpočtů odesílá signál v kanále `cmpDone` značící, že porovnávání bylo dokončeno, a vrací se do počáteční lokace.



Obrázek 4.4: Časový automat reprezentující model kontroloru rovnosti výsledků

Model synchronizace vstupních hodnot

Model čeká na vygenerování nových vstupních hodnot (tedy na signál `update?`). Poté pomocí kanálu `change[idx]` postupně dává signál jednotlivým logickým členům, že byl jejich vstupní bit aktualizován. Proměnná `idx` představuje pozici jednotlivých bitů. Model iteruje mezi dvěma zavázanými lokacemi tak dlouho, dokud se neprojde přes všechny vstupní bity (konstantní proměnná `NPI` značí počet vstupních bitů násobičky). Po aktualizování všech bitů se model vrací do počáteční lokace.



Obrázek 4.5: Časový automat reprezentující model zajišťující synchronizaci vstupních hodnot

Model logického hradla

Modeluje logický člen se dvěma vstupy a jedním výstupem. Model při inicializaci přijímá následující vstupní parametry:

```

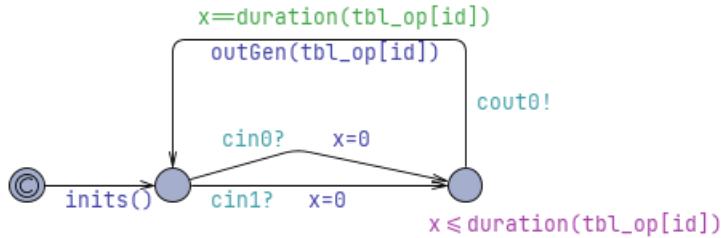
const int id, const int a0, const int a1, const int y0,
broadcast chan &cin0, broadcast chan &cin1, broadcast chan &cout0,
  
```

kde konstanta `id` je jedinečný identifikátor daného hradla. Konstatní proměnné `a0` a `a1` značí pozice vstupních bitů v globálním poli `bits`, konstanta `y0` je pozice výstupního bitu ve stejném poli. Broadcastové synchronizační kanály `cin1` a `cin2` slouží k přijetí signálu o tom, že byl jeden ze vstupních bitů aktualizován (jedná se tedy o kanál `change` zmínený výše u modelu synchronizace). Přes synchronizační kanál `cout0` rozesílá hradlo signál

o aktualizaci svého výstupního bitu. Tento signál poté přijímají všechna další hradla, která používají jeho výstupní bit jako svůj vstupní bit.

Po inicializaci model čeká na aktualizace jednoho ze vstupních bitů (kanály `cin0` a `cin1`). Poté dochází k vygenerování výstupu (`outGen(tbl_op[id])`) a po uběhnutí zpoždění signálu (`x == duration(tbl_op[id])`) se model vrací zpět do lokace čekání na nové vstupy.

Hradla mohou vykonávat následující logické operace: AND, NAND, OR, NOR, XOR a XNOR. Každé hradlo má svoji operaci uloženou v poli `tbl_op` na pozici `id`. Zpoždění jednotlivých operací se liší, pro každou z nich je definováno v rámci globální funkce `duration`.



Obrázek 4.6: Časový automat reprezentující model logického hradla

4.1.3 Inicializace systému

V závěru dochází k inicializaci a propojení celého systému. Ve výpisu 4.2 je vidět inicializace jednotlivých součástí. `synPri` představuje model synchronizace, `ediff` zase model kontroly rovnosti. `mul2A` je model přesné násobičky, `mul2Atb` představuje generátor náhodných vstupních hodnot. Na konci lze pozorovat inicializaci jednotlivých logických hradel `gate2`.

```

1 synPri = syncPrimary();
2
3 mul2A = tmul2any(PIxy, P0x, tbl_acc_any, DLY_MUL2);
4 mul2Atb = tmul2_tb_random(DLY_MUL2, COVERAGE_RATIO);
5
6 ediff = eval_diff(5);
7
8 //gates
9 g122=gate2(0, PIxy[14], PIxy[0], 122, change[14], change[0], change[122]);
10 g123=gate2(1, PIxy[15], PIxy[0], 123, change[15], change[0], change[123]);
11 g127=gate2(2, PIxy[10], PIxy[4], 127, change[10], change[4], change[127]);
12 g129=gate2(3, PIxy[13], PIxy[1], 129, change[13], change[1], change[129]);
13 g130=gate2(4, PIxy[14], PIxy[1], 130, change[14], change[1], change[130]);
14 g133=gate2(5, PIxy[15], PIxy[1], 133, change[15], change[1], change[133]);
15 g142=gate2(6, 122, 129, 142, change[122], change[129], change[142]);
16 //...
17 //...
18 //...
19 g433=gate2(214, 431, 430, 433, change[431], change[430], change[433]);
20 g434=gate2(215, 431, 430, P0y[14], change[431], change[430], change[434]);
21 g435=gate2(216, 432, 433, P0y[15], change[432], change[433], change[435]);
22 g436=gate2(217, 318, 210, P0y[3], change[318], change[210], change[436]);
  
```

Výpis 4.2: Inicializace systému

4.2 Převod modelů z jazyka Verilog do modelů v nástroji UPPAAL

Jak již bylo zmíněno, zdrojem modelů přibližných bezznaménkových násobiček je knihovna EvoApproxLib [21]. K překladu těchto modelů do modelů v prostředí UPPAAL byl implementován skript v jazyce Python nazvaný `parse.py`.

Skript je schopen přeložit modely z jazyka Verilog ve formátu zobrazeném ve výpisu 4.3. Zvládne tedy přeložit všechny modely násobiček 7x7, 8x8 (kromě násobičky `mul8u_1JJQ`) a 11x11 bitů. Modely násobiček 12x12 a 16x16 bitů mají odlišnou (složitější) strukturu. Jsou modelovány pomocí skládání menších přibližných násobiček do sebe a skript `parse.py` toto neimplementuje.

```
1 module mult_name (
2     A,
3     B,
4     O~);
5
6     input [n:0] A;
7     input [n:0] B;
8     output [m:0] O;
9
10    wire sig_10,sig_11, ...;
11
12    assign sig_10 = A[0] & B[1];
13    ...
14    assign sig_30 = sig_15 & sig_20;
15    ...
16
17    assign O[15] = sig[30];
18    ...
19    assign O[0] = 1'b0;
```

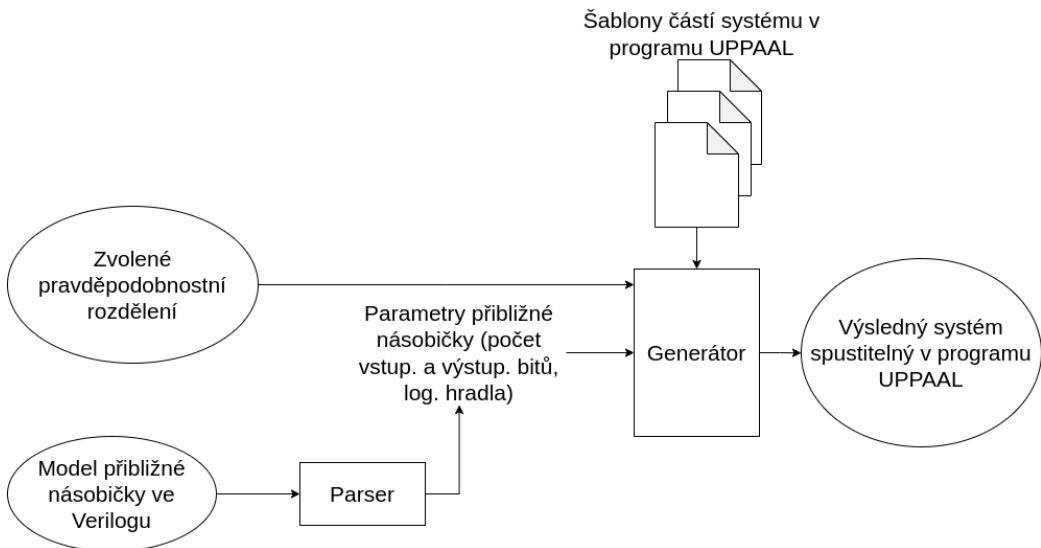
Výpis 4.3: Příklad platného formátu modelu násobičky v jazyce Verilog

Základní princip fungování skriptu

Základní princip fungování skriptu `parse.py` je ilustrován na obrázku 4.7.

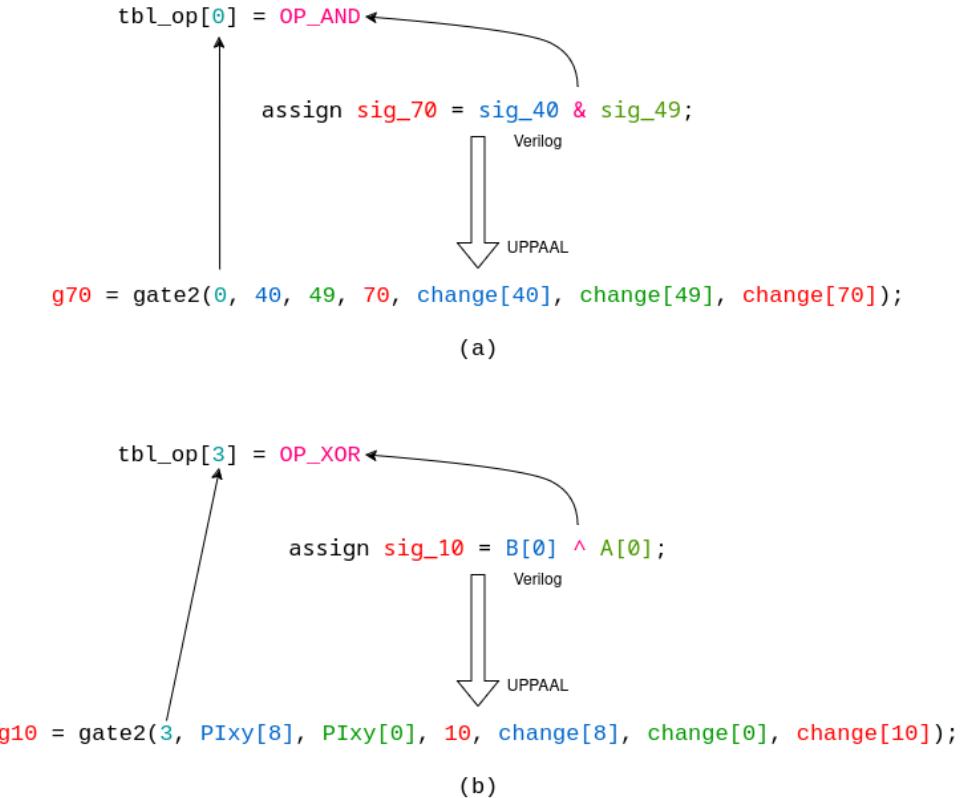
Prvním krokem je otevření a načtení vstupního souboru napsaného ve Verilogu. Při první iteraci přes jednotlivé řádky souboru je získána informace o počtu vstupních a výstupních bitů násobičky. Ta je využita k nastavení některých vnitřních parametrů skriptu a také k pozdějšímu vygenerování globálních proměnných `NPI`, `NPO` a polí `PIxy`, `P0x` a `P0y`.

Při druhé iteraci přes jednotlivé řádky vstupního souboru dochází k vytváření instancí logických hradel. Na obrázku 4.8 je vidět princip převodu logické operace dvou signálů ze vstupního souboru na instanci hradla `gate2` v prostředí UPPAAL. Části, které se na sebe vzájemně mapují, jsou odlišeny barevně. Také je vidět uložení typu logické operace do tabulky `tbl_op`. ID hradla, které se používá jako index do tabulky (v tomto příkladě hodnota 0), je s každým hradlem postupně inkrementováno a začíná vždy od 0 bez ohledu na další hodnoty.



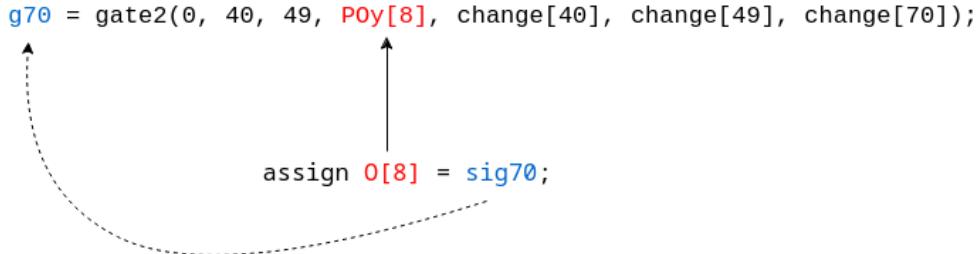
Obrázek 4.7: Schéma fungování skriptu `parse.py`

Na obrázku 4.8 lze pozorovat dvě varianty parametrů. V části (a) se jedná o logické hradlo, které přebírá výsledky operací z předchozích hradel. Hradlo v části (b) má na vstupu přímo vstupní bity násobičky $B[0]$ a $A[0]$. Dále se v modelech vyskytuje různé kombinace těchto a dalších variant, princip jejich překladu ovšem zůstává stejný.



Obrázek 4.8: Vytvoření instance logického hradla v nástroji UPPAAL z předlohy ve Verilogu

Na konci vstupního souboru dochází k přiřazování jednotlivých signálů k výstupním bitům. To je ve skriptu řešeno tak, že se podle čísla v názvu signálu vyhledá hradlo se stejným číslem v názvu a v jeho parametrech se aktualizuje index výstupního bitu na index $P0y[n]$ značící index výstupního bitu celé přibližné násobičky (viz obrázek 4.9).



Obrázek 4.9: Aktualizace výstupního bitu logického hradla

V závěru skriptu poté dochází k vygenerování jednotlivých částí výstupního souboru. Soubory systémů v programu UPPAAL jsou napsány v jazyce XML, neboť jejich součástí jsou i grafické objekty (modely časových automatů) definované pomocí uzlů XML. Vygenerované části jsou nakonec spojeny a uloženy do výstupního souboru, s nímž je poté možné pracovat v programu UPPAAL.

Skript lze spouštět s přepínačem `--distribution [DISTRIBUTION]`, který nastavuje vybrané pravděpodobnostní rozdělení pro generování náhodných vstupů. Při vytváření modelu generátoru náhodných vstupů je poté vybrané rozdělení zohledněno. Rozdělení jsou ve skriptu připravena pro násobičky 8x8 bitů (tedy generují čísla 0 až 255). Pro jiné typy násobiček by bylo třeba tyto hodnoty změnit, případně je lze změnit po vygenerování ve finálním souboru v lokálních deklaracích modelu `tmul2_tb_random`.

Pseudokód struktury funkce `main` skriptu `parse.py` je k vidění ve výpisu 4.4.

```

1 read(input_file)
2
3 for line in input_file:
4     parse_input_output(line)
5
6 create_PIxy_mapping()
7 create_P0y_mapping()
8
9 for line in input_file:
10    parse_gates(line)
11
12 generate_parts()
13 output_file = join_parts()
14
15 write(output_file)
  
```

Výpis 4.4: Pseudokód hlavní části skriptu `parse.py`

4.3 Zvolené aplikace

Klíčovou myšlenkou práce bylo ověření vlastností přibližných násobiček v závislosti na různých pravděpodobnostních rozděleních jejich vstupů. V rámci této práce bylo zvoleno několik algoritmů, u nichž dochází k násobení celých kladných čísel. Každý algoritmus byl

s vhodnými vstupy simuloval s tím, že se postupně sbírala data o dvojicích násobených čísel. Tato data byla poté vizualizována a následně byl proveden odhad pravděpodobnostního rozdělení (nebo kombinace více rozdělení), které by přibližně odpovídalo danému naměřenému rozdělení.

Výstupy vizualizací, z nich odvozené odhady pravděpodobnostních rozdělení a pseudo-kódy ilustrující generování těchto rozdělení jsou uvedeny v příloze D. Zde následuje pouze stručný výpis vybraných aplikací a od nich odvozených rozdělení:

- **Algoritmus Ellipse Mid-Point** – vstup X je generován pomocí rozdělení beta s parametry $\alpha = 0,5$ a $\beta = 5$. Vstup Y je generován dle mírně deformovaného rovnoměrného rozdělení od 0 do 255. Zvolené rozdělení bylo pojmenováno **beta_uni**.
- **Bresenhamův rasterizační algoritmus** – jako vstup X byla zvolena konstanta 2. Vstup Y je generován s normálním rozdělením se střední hodnotou 150 a rozptylem 30. Zvolené rozdělení bylo pojmenováno jako **const_norm**.
- **Pritchardovo síto** – vstup X je generován dle rozdělení gamma s parametry $k = 1,0$ a $\theta = 0,8$ a poté vynásoben hodnotou 7. Vstup Y je generován pomocí dvou normálních rozdělení okolo středů 50 a 220. Rozdělení bylo pojmenováno jako **gamma_2norm**.
- **Algoritmus Integer Square Root** – vstupy X a Y mají vždy stejnou hodnotu. Jsou generovány pomocí triangulárního rozdělení s minimem v -10, maximem v 255 a modelem 10. Zvolené rozdělení bylo pojmenováno jako **same_triang**.
- **Algoritmus Circle Point to Point** – vstupy X a Y mají opět vždy stejnou hodnotu. Jsou generovány s rovnoměrným rozdělením v intervalu $\langle 0, 255 \rangle$. Zvolené rozdělení bylo pojmenováno jako **same_uni**.
- **Algoritmus AKS** – vstup X je generován s triangulárním rozdělením s minimem v -50, maximem ve 450 a modelem 70. Vstup Y je generován dle rozdělení beta s parametry $\alpha = 2$ a $\beta = 2$, následně je vynásoben hodnotou 255. Zvolené rozdělení bylo pojmenováno jako **triang_beta**.
- **Algoritmus ElGamal** – vstup X je generován dle deformovaného triangulárního rozdělení s minimem a modelem 0 a maximem 350. Vstup Y je generován s Weibullovým rozdělením s parametry $k = 1,7$ a $\lambda = 1,7$. Zvolené rozdělení bylo pojmenováno jako **triang_weibull**.

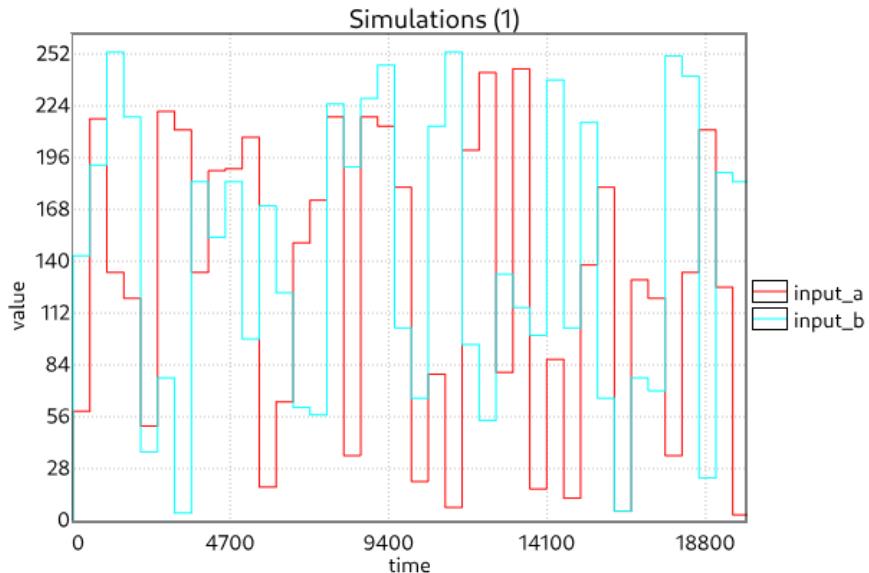
4.4 Simulační dotazy

Pro simulaci chování násobiček bylo v systému UPPAAL vytvořeno několik dotazů.

První z nich slouží k ověření, že se správně generují vstupní hodnoty. Zároveň může čtenáři sloužit k upřesnění představy o tom, jaké vstupy se generují v rámci jednotlivých rozdělení. Výstupem je graf znázorňující hodnoty vstupních čísel v jednotlivých časových okamžicích.

Jeden příklad výstupu lze pozorovat na obrázku 4.10, příklady pro všechny typy použitých rozdělení jsou poté uvedeny v příloze E v části **Kontrola vstupních hodnot**. Dotaz vypadá následovně:

```
simulate[<=20000;1] input_a, input_b
```



Obrázek 4.10: Příklad výstupu dotazu ověřujícího hodnoty vygenerovaných vstupů

Druhý dotaz slouží k porovnání výstupů přesné a zkoumané přibližné násobičky. Podobně jako u předchozího dotazu je výstupem graf zobrazující obě hodnoty v jednotlivých časových okamžicích. Příklady výstupů pro násobičky s různou úrovní aproximace lze pozorovat v příloze E v části **Porovnání přesných a přibližných výstupů**. Dotaz má následující podobu:

```
simulate[<=20000;1] res_acc, res_approx
```

Třetí dotaz zobrazuje zpoždění, které zkoumaná násobička nabírá při jednotlivých výpočtech. To může sloužit jednak k porovnání zpoždění jednotlivých násobiček, dále lze z grafů i vypozorovat, kolika dílčími hradly daný výpočet prochází. Příklady výstupů pro méně přesnou a velmi přesnou approximační násobičku lze pozorovat v příloze E v části **Sledování zpoždění**. Dotaz vypadá následovně:

```
simulate[<=1000;1] current_delay
```

Čtvrtý dotaz tvoří hlavní část experimentů. V rámci tohoto dotazu jsou sledovány všechny vybrané hodnotící metriky. Dotaz má následující podobu:

```
simulate[<=10000000;1] {coverage_percentage, delay_avg, error_prob,
mean_abs_error, mean_relative_error, mean_squared_error,
avg_flips_per_res, worst_case_error, worst_case_relative_error,
max_hamming_distance, max_bit_flips, worst_delay}
```

Význam jednotlivých sledovaných proměnných je popsán v tabulce 4.1.

Většina těchto metrik byla již popsána v sekci 2.3. Překlopení bitu logického hradla je situace, v níž se výstupní bit daného hradla po výpočtu nastaví na opačnou hodnotu, než

Název	Význam
coverage_percentage	procento pokrytí všech kombinací vstupů
delay_avg	průměrné zpoždění jednoho výpočtu přibližně násobičky
error_prob	pravděpodobnost chyby
mean_abs_error	průměrná absolutní chyba
mean_relative_error	průměrná relativní chyba
mean_squared_error	průměrná kvadratická chyba
avg_flips_per_res	průměrný počet překlopených bitů v logických hradlech v rámci jednoho výpočtu
worst_case_error	nejhorší absolutní chyba
worst_case_relative_error	nejhorší relativní chyba
max_hamming_distance	maximální Hammingova vzdálenost mezi výsledky přesné a přibližné násobičky
max_bit_flips	maximální počet překlopených bitů v logických hradlech v rámci jednoho výpočtu
worst_delay	nejhorší zpoždění jednoho výpočtu přibližně násobičky

Tabulka 4.1: Význam sledovaných proměnných

jakou měl před výpočtem. Jedná se o metriku, kterou lze dále využít k výpočtu energetické spotřeby obvodu.

Doba trvání běhu 10 000 000 časových jednotek byla určena experimentálně. Za tuto dobu je systém schopen pokrýt přibližně čtvrtinu všech vstupních kombinací (u referenčního rozdělení `uni_uni`), což zajišťuje dostatečně přesné výsledky ve srovnání s referenční knihovnou `EvoApproxLib`.

Výstupem simulace je soubor ve formátu csv, ze kterého jsou pomocí skriptu `process_results.py` extrahována veškerá relevantní data. Ta jsou poté uložena do struktury Pandas Dataframe a do tzv. *pickle* souboru (soubor s příponou .pkl vytvořený modulem `pickle`).

Alternativním způsobem řešení by mohly být dotazy typu

`E[<=10000000; 1] (max: error_prob),`

které by byly volány pro každou metriku zvlášť. Tím by se ovšem značně zvýšila časová náročnost simulací (pro každou násobičku v řádech desítek minut až jednotek hodin, v závislosti na výkonu počítače a také na počtu hradel dané násobičky).

4.4.1 Možná rozšíření o další dotazy

Práci by šlo dále rozšířit o dotazy na odhad pravděpodobnosti. Například odhad, s jakou pravděpodobností se za určitou dobu eventuelně dostane procento pokrytí všech vstupních kombinací alespoň na danou hodnotu (v příkladu 50 %):

`Pr[<=10000000] (<> coverage_percentage > 50).`

Dalším příkladem by mohl být odhad toho, s jakou pravděpodobností bude celkový počet překlopení bitů v jednotlivých logických hradlech za určitou dobu menší než daná hodnota:

`Pr[<=10000000] ([] bit_flips_sum < 10000).`

Kapitola 5

Výsledky experimentů

Z knihovny EvoApproxLib bylo vybráno celkem 14 přibližných násobiček 8x8 bitů, na kterých byly prováděny výše popsané simulační dotažy. Při volbě jednotlivých násobiček byla zohledněna jejich plocha a z toho plynoucí úroveň aproximace. Ve vybrané sadě se tak objevují jak násobičky, které jsou sestaveny z méně než 10 logických hradel, tak naopak téměř přesné approximační násobičky složené z několika stovek logických hradel.

V referenční tabulce 5.1 jsou vypsány všechny zvolené násobičky spolu s některými sledovanými metrikami. Údaje o velikosti plochy obvodů byly převzaty z knihovny EvoApproxLib [21]. Ostatní data pocházejí z výsledků simulací s referenčním rozdelením generovaných vstupů **uni_uni**. V tabulce jsou pro porovnání zahrnutы také přesné hodnoty metrik z knihovny EvoApproxLib. Odchylky naměřených hodnot oproti hodnotám z knihovny jsou způsobeny tím, že v rámci simulací nedocházelo z časových důvodů k pokrytí celé množiny vstupních kombinací.

Násobička	Plocha	Pravděpodobnost chyby		Průměrná absolutní chyba		Nejhorší absolutní chyba	
		měření	EvoApproxLib	měření	EvoApproxLib	měření	EvoApproxLib
mul8u_17MN	13.1	0.97	0.99	2342.61	5249	17726	17853
mul8u_17MJ	18.8	0.89	0.99	1406.84	4276	17030	17793
mul8u_R36	60.5	0.98	0.98	3764.64	3828	32289	32289
mul8u_Z9D	220.6	0.89	0.89	3141.56	3169	31752	32258
mul8u_17R6	228.5	0.98	0.99	185.57	442	1905	1925
mul8u_2NDH	347.8	0.99	0.99	93.52	285	2709	2709
mul8u_197B	395.6	0.98	0.98	69.92	119	424	431
mul8u_NLX	511.5	0.95	0.96	6.06	42	161	161
mul8u_GTR	550.5	0.84	0.85	20.1	29	125	125
mul8u_BG1	561.8	0.49	0.5	289.01	284	2932	2938
mul8u_R92	604.5	0.83	0.88	2.84	11	39	40
mul8u_ZB3	682.8	0.07	0.06	0.14	0.12	2	2
mul8u_12KA	683.3	0.09	0.09	11.91	12	192.0	192

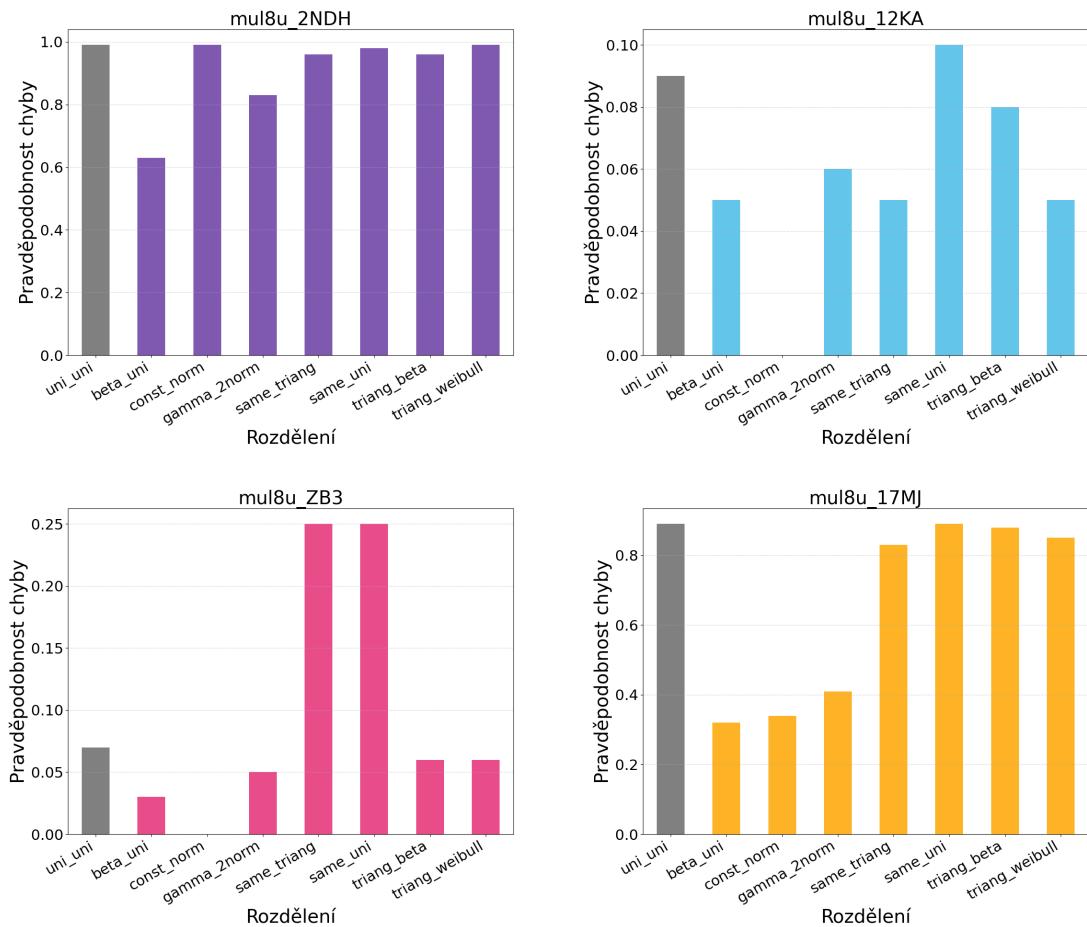
Tabulka 5.1: Zkoumané násobičky a vybrané metriky naměřené s rozdelením **uni_uni** vs přesné hodnoty z EvoApproxLib

V rámci této kapitoly je prezentován výběr z výsledků v podobě sloupcových grafů vždy ve skupinách po čtyřech. Každá skupina se váže k jedné hodnotící metrice, každý graf potom prezentuje výsledky jedné konkrétní násobičky. Jednotlivé sloupce představují různá pravděpodobnostní rozdělení použitá při generování vstupů (viz sekce 4.3 a příloha D), referenční rozdelení **uni_uni** má ve všech grafech šedou barvu. Kompletní výsledky

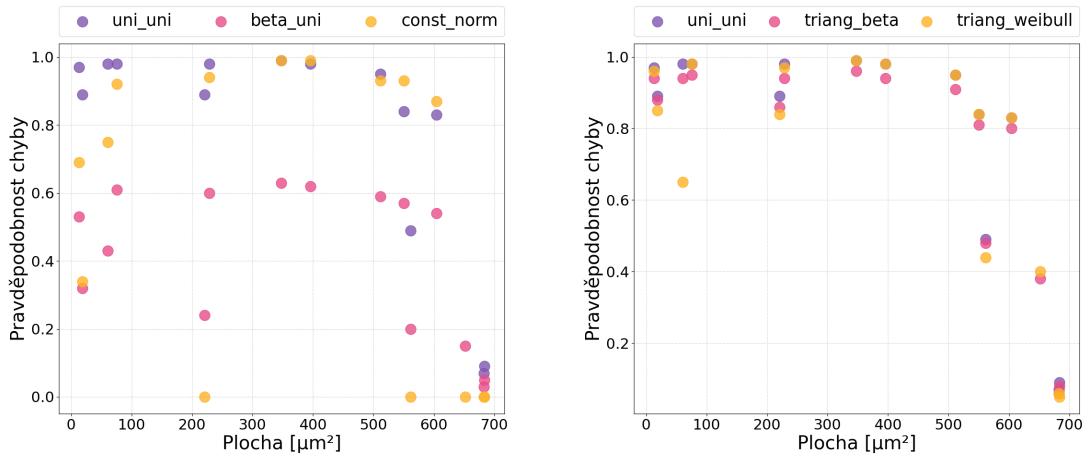
simulací všech násobiček v rámci všech rozdělení jsou k dispozici v příloze F, každá tabulka se vztahuje k jednomu danému rozdělení.

Pravděpodobnost chyby u většiny zkoumaných násobiček buď zůstávala podobná, jako u referenčního rozdělení, nebo o něco klesla. K výraznějšímu poklesu pravděpodobnosti chyby docházelo zpravidla u rozdělení, kde alespoň jedno z generovaných čísel nabývá nízkých hodnot (rozdělení `beta_uni`, `const_norm` a `gamma_2norm`). U rozdělení `const_norm`, kde je jeden ze vstupů vždy konstanta 2, dokonce některé násobičky generovaly 100 % přesné výsledky, tedy pravděpodobnost chyby byla nulová.

Zajímavou anomálií byla násobička `mul8u_ZB3`, u které došlo k výraznému zvýšení pravděpodobnosti chyby u rozdělení, která modelovala umocňování čísel na druhou, tedy kde byly oba vstupy stejné (rozdělení `same_triang` a `same_weibull`). Grafy pravděpodobnosti chyby některých vybraných násobiček lze pozorovat na obrázku 5.1. Bodové grafy v obrázku 5.2 potom nabízejí srovnání pravděpodobnosti chyby násobiček v rámci jednotlivých pravděpodobnostních rozdělení.



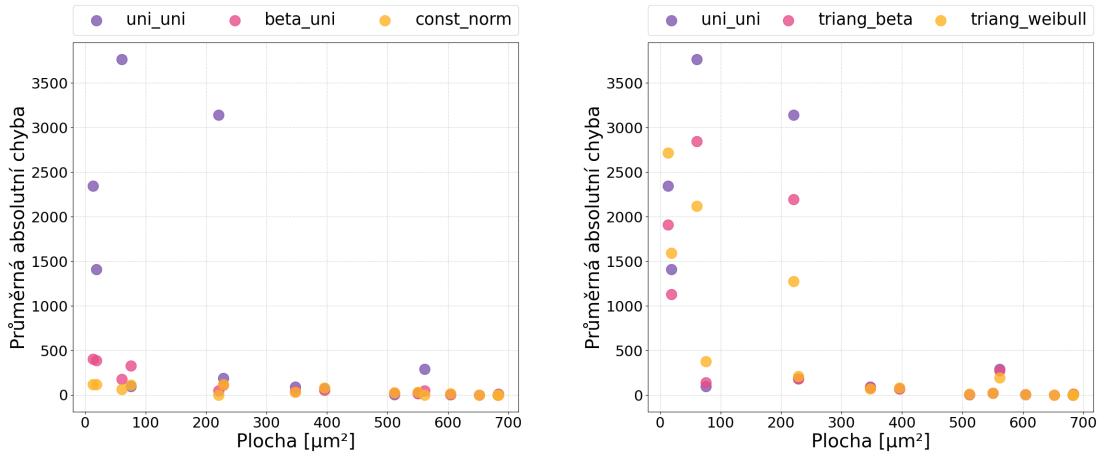
Obrázek 5.1: Pravděpodobnost chyby vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot



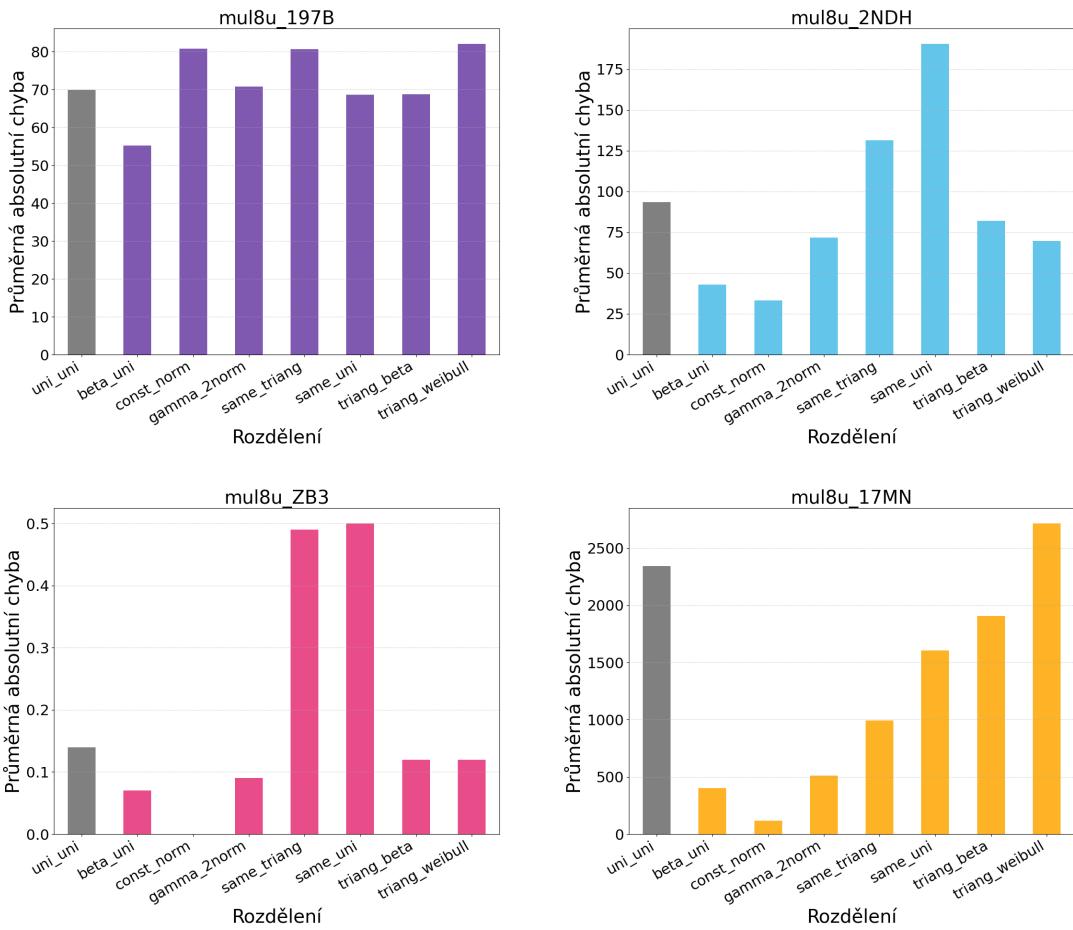
Obrázek 5.2: Porovnání pravděpodobnosti chyby sledovaných násobiček v rámci vybraných rozdělení

Průměrná absolutní chyba kolísala v závislosti na úrovni aproximace jednotlivých násobiček. U velmi přesných z nich se naměřené hodnoty pohybovaly nejčastěji v řádech desítek či jednotek (někdy i desetin). Oproti tomu u méně přesných násobiček mnohdy docházelo ke zlepšení (méně často také ke zhoršení) hodnoty průměrné absolutní chyby o několik stovek či tisíc.

Míra zlepšení v rámci jednotlivých pravděpodobnostních rozdělení byla poměrně variabilní. K o něco výraznějším zlepšením opět docházelo u rozdělení generujících menší hodnoty. Výsledky simulací vybraných násobiček jsou vidět v grafech na obrázku 5.4. Obrázek 5.3 poté opět slouží k porovnání průměrné absolutní chyby napříč vybranými pravděpodobnostními rozděleními.

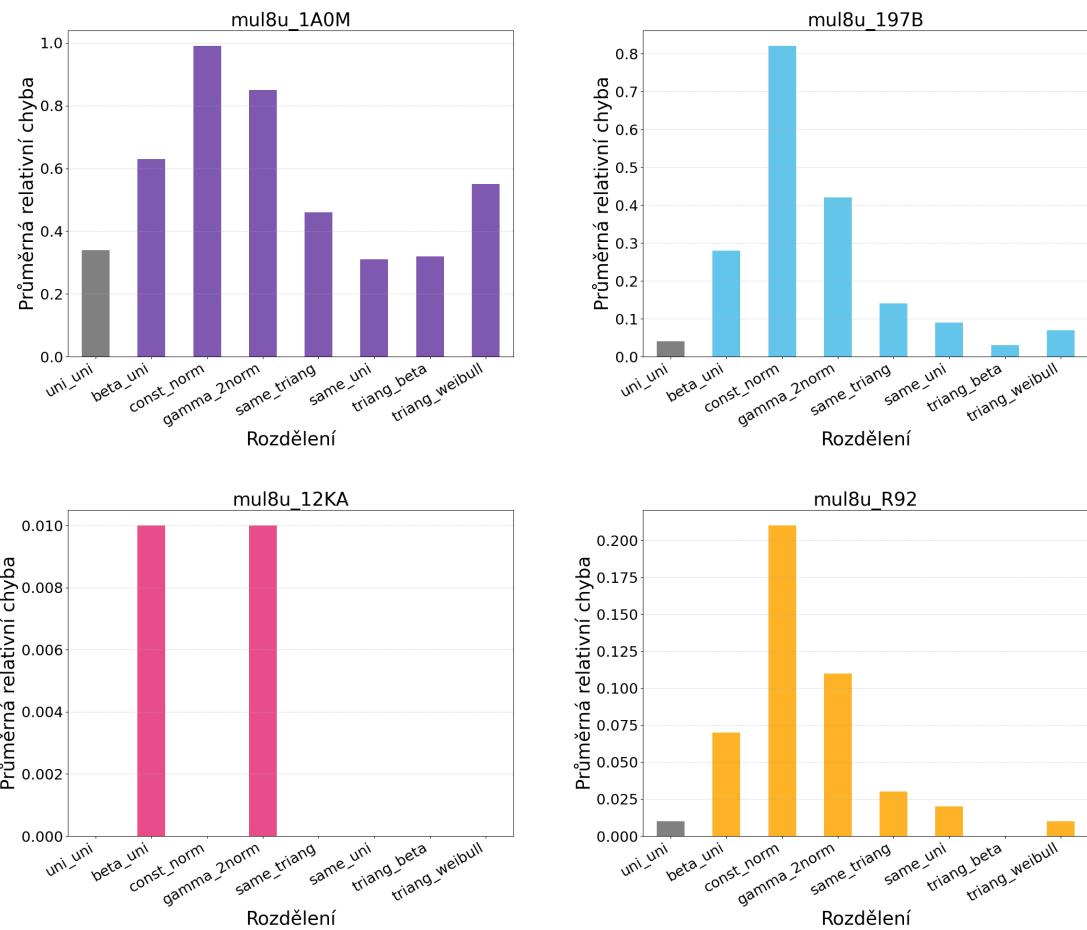


Obrázek 5.3: Porovnání průměrné absolutní chyby sledovaných násobiček v rámci vybraných rozdělení

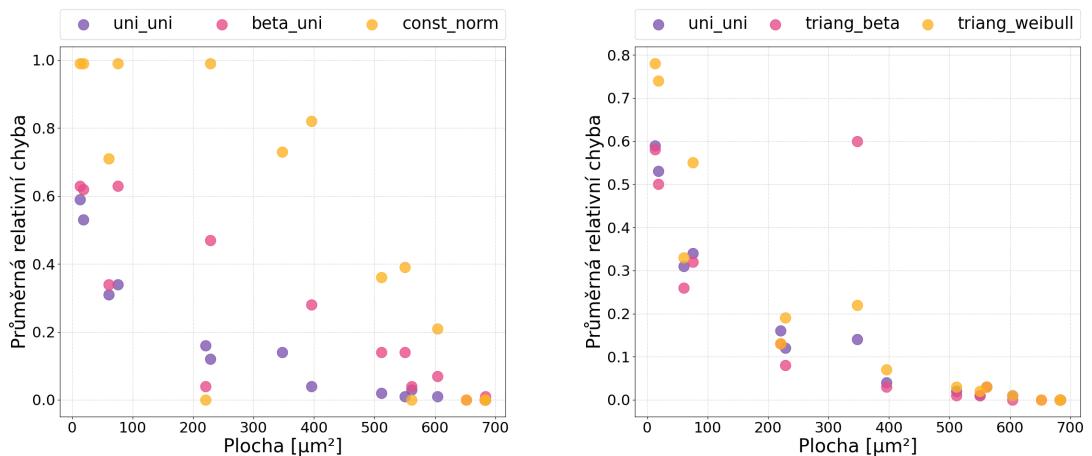


Obrázek 5.4: Průměrná absolutní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot

Průměrná relativní chyba udává, jak velká chyba vznikla v porovnání s očekávaným výsledkem. Pokud měl například výsledek výpočtu být 3000 a výstup přibližného obvodu byl 2000, pak je relativní chyba $1/3$. V rámci výsledků simulací se větší relativní chyba vyskytovala u rozdělení, která generovala menší vstupy. To není překvapivé, neboť u malých výsledků se každá odchylka projeví ve výpočtu relativní chyby více, než je tomu u větších výsledků. Příklady relativní chyby vybraných násobiček lze pozorovat na obrázku 5.5. Na obrázku 5.6 je vidět porovnání průměrné relativní chyby jednotlivých násobiček v rámci vybraných pravděpodobnostních rozdělení.

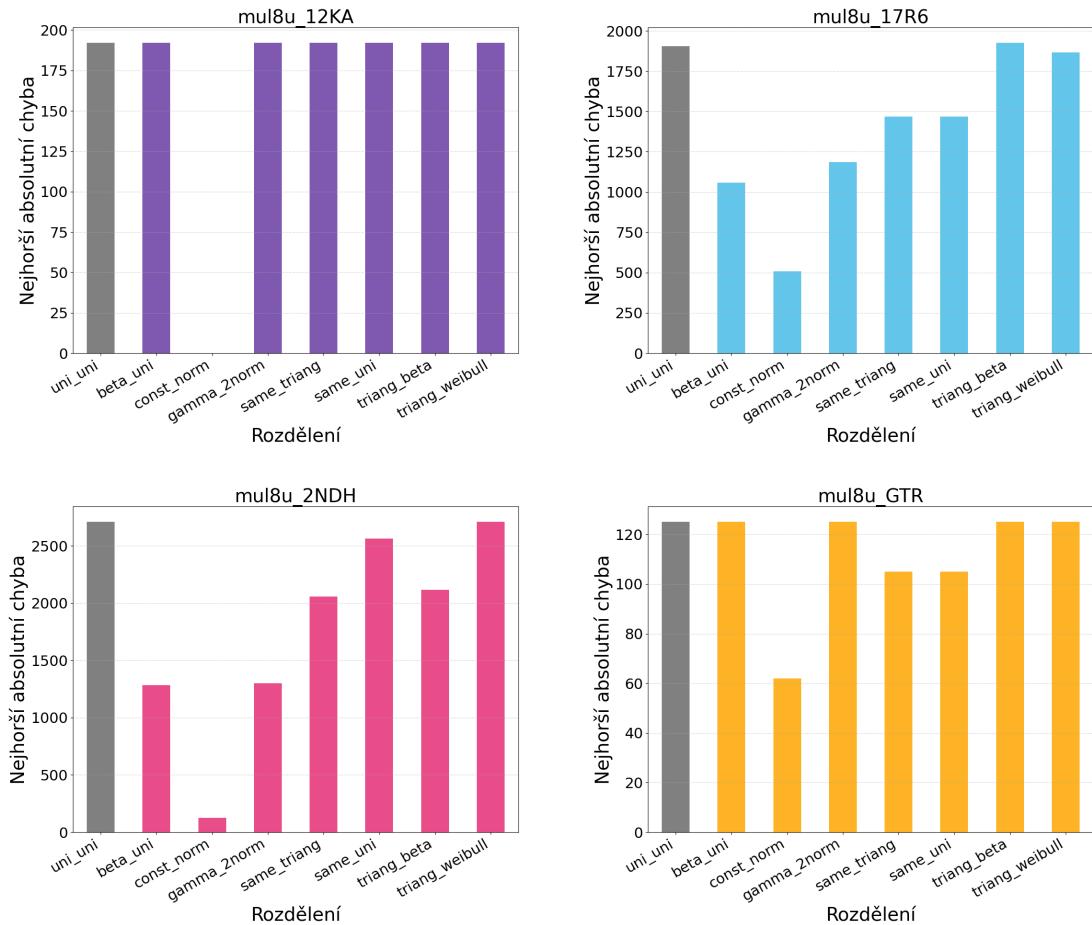


Obrázek 5.5: Průměrná relativní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot



Obrázek 5.6: Porovnání průměrné relativní chyby sledovaných násobiček v rámci vybraných rozdělení

Nejhorší absolutní chyba bývala zpravidla nižší u rozdělení s menšími vstupy, počátkem výstupy. Míra tohoto typu chyby by nikdy neměla přesáhnout míru chyby naměřené při generování vstupů s referenčním rozdělením `uni_uni`, neboť u tohoto rozdělení by v ideálném případě došlo k vygenerování všech vstupních kombinací. Při simulacích z časových úspor nedochází ke 100% pokrytí všech vstupních kombinací, proto se občas může u jiného rozdělení vyskytnout mírně větší nejhorší absolutní chyba.



Obrázek 5.7: Nejhorší absolutní chyba vybraných násobiček pro jednotlivá pravděpodobnostní rozdělení vstupních hodnot

Kapitola 6

Diskuze škálovatelnosti

Tato kapitola se zabývá škálovatelností modelovaného systému s ohledem na paměťovou a časovou náročnost. Simulace byly prováděny na počítači s následujícími parametry:

- OS: Fedora Linux 39 (Workstation Edition),
- procesor: Intel Core i5-13600KF,
- paměť: 32 GB,
- verze software: UPPAAL 5.0.

Program UPPAAL se při simulacích nachází v následujícím nastavení:

- Search Order: Breadth First,
- Exploration: Exhaustive (Symbolic),
- State Space Reduction: Conservative,
- State Space Representation: Compact Data Structure,
- Diagnostic Trace: None,
- Back-propagation Order: Depth First,
- Search Priority: Automatic,
- Strategy: Entire winning strategy over all states,
- Extrapolation: Automatic,
- Hash Table Size: 16 MB,
- Learning Filter: Local Filter,
- Learning Method: Q-learning.

V tabulce 6.1 jsou uvedeny výsledky měření pro vybrané násobičky. Násobičky z knihovny EvoApproxLib byly vybrány s ohledem na různý počet vstupních bitů a také na různý počet logických hradel.

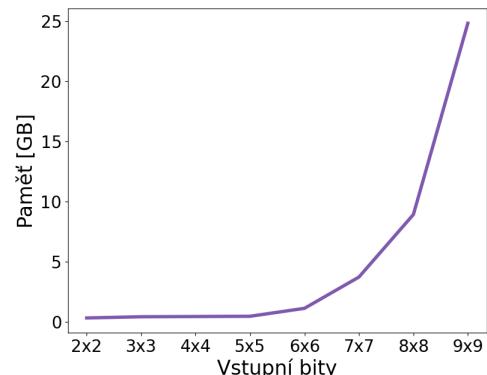
Násobička	Vstupní bity	Paměť [GB]	Čas [s]	Počet hradel
mul2u	2x2	0.3	1	5
mul7u_09Y	7x7	3.7	75	73
mul7u_073	7x7	4.3	760	154
mul7u_01Q	7x7	4.3	670	209
mul8u_17MN	8x8	8.9	85	10
mul8u_R36	8x8	8.9	50	30
mul8u_17R6	8x8	8.9	255	88
mul8u_197B	8x8	8.9	550	175
mul8u_GTR	8x8	8.9	1005	241
mul8u_12KA	8x8	8.9	1540	319

Tabulka 6.1: Výsledky měření časové a paměťové náročnosti

Jak je z výsledků patrné, paměťová náročnost závisí převážně na počtu vstupních bitů násobičky. Z toho důvodu bylo možné vyrobit násobičky s jinými vstupními bity, než jaké jsou k dispozici v knihovně EvoApproxLib. Toho bylo docíleno úpravou vybraných násobiček z knihovny – odebráním přebytečných vstupních a výstupních bitů a signálů s nimi spojených. Na těchto násobičkách bylo poté opět provedeno měření s ohledem pouze na paměťovou složitost.

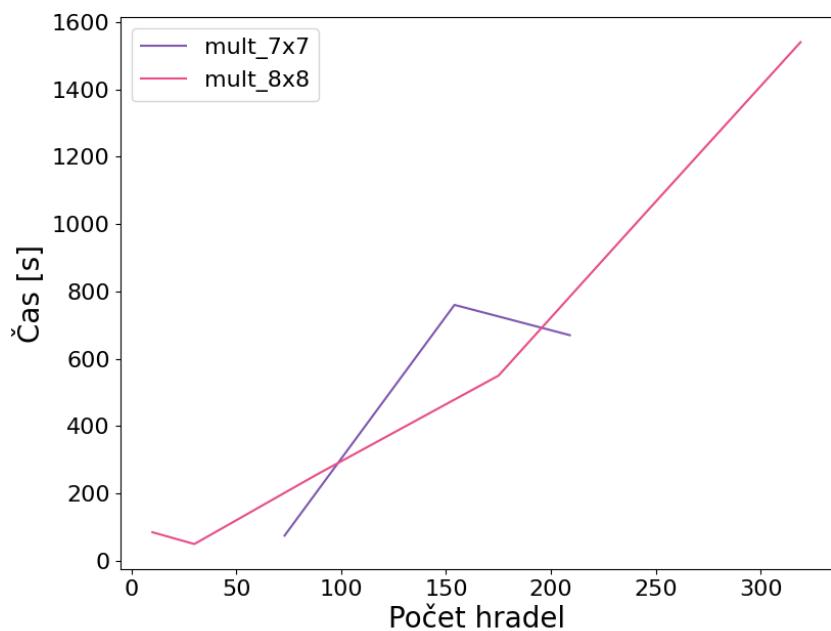
Výsledky lze pozorovat v tabulce 6.2 a na obrázku 6.1. Dochází zde k exponenciálnímu růstu pravděpodobně z toho důvodu, že program UPPAAL při ověřování systému prochází velkou část stavového prostoru (ne nutně celý stavový prostor), který se s každým zvětšením násobičky o 2 výstupní bity zvětšuje čtyřikrát. Na násobičku 10x10 bitů již paměť referenčního stroje nestačila, proto lze pouze konstatovat, že paměťová náročnost této násobičky je větší než 32 GB.

Vstupní bity	Paměť [GB]
2x2	0.3
3x3	0.4
4x4	0.42
5x5	0.44
6x6	1.1
7x7	3.7
8x8	8.9
9x9	24.8
10x10	>32



Tabulka 6.2: Paměťová náročnost dle počtu
Obrázek 6.1: Paměťová náročnost dle po-
vstupních bitů

Časová náročnost simulací je lineárně závislá na počtu logických hradel v násobičce. Dále se časová náročnost odvíjí od výkonu procesoru počítače, na kterém jsou simulace prováděny. Na obrázku 6.2 je vidět časová náročnost v závislosti na počtu hradel u násobiček 7x7 a 8x8 bitů. Z grafu také plyne, že počet vstupních bitů násobičky nemá na časovou složitost simulace výrazný vliv.



Obrázek 6.2: Časová náročnost simulací v závislosti na počtu logických hradel

Kapitola 7

Závěr

Cílem této bakalářské práce bylo porovnání vlastností vybraných přibližných násobiček v rámci kontextu různých aplikací. Modely jednotlivých násobiček byly převzaty z knihovny EvoApproxLib [21] a následně převedeny do modelů v simulačním nástroji UPPAAL. V rámci tohoto prostředí byly poté prováděny simulace jednotlivých modelů násobiček s různými pravděpodobnostními rozdeleními při generování jejich vstupů. Při simulování byly sledovány především chybové metriky jako pravděpodobnost chyby, průměrná absolutní chyba aj.

Z výsledků lze usuzovat, že různá rozdelení vstupů mají na chybovost výsledků výpočtu značný vliv. To bylo obzvláště patrné u méně přesných násobiček, u kterých často chybové metriky kolísaly oběma směry v závislosti na daném rozdelení. Z toho plyne, že pokud by se násobička, která je v případě testování všech kombinací vstupů velmi nepřesná, použila v určitém vhodném kontextu, mohlo by být dosaženo výrazného snížení energetické spotřeby a časové náročnosti výpočtu se zachováním dostatečné přesnosti. Na druhou stranu při použití stejných násobiček v nevhodném kontextu by mohla chybovost výpočtu být výrazně horší.

Je nutno podotknout, že výsledky simulací nejsou 100% přesné, neboť v rámci simulací z časových důvodů nedocházelo k vyzkoušení všech kombinací možných vstupních hodnot. Zde proto vidíme první možnost pro vylepšení práce.

Další rozšíření bylo možné učinit přizpůsobením skriptu `parse.py`, který převádí modely z EvoApproxLib do UPPAAL, dalším modelům z této knihovny. Momentálně je uzpůsoben pro beznaménkové násobičky 7x7, 8x8 a 11x11 bitů. Poměrně jednoduše bylo možné skript rozšířit i pro aproximační sčítáčky nebo pro násobičky s rozdílně velkými vstupy.

Také by bylo možné sledovat některé složitější hodnotící metriky. Sledovanou metriku o počtu překlopení bitů bylo možné například využít k výpočtu energetické spotřeby obvodu.

Literatura

- [1] AGARWAL, A., RINARD, M., SIDIROGLOU, S., MISAILOVIC, S. a HOFFMANN, H. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures. *MIT-CSAIL-Tech.rep.-2009-042*. Září 2009.
- [2] ANDRÉS, M. E. Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems. *CoRR*. 2011, abs/1111.2760. Dostupné z: <http://arxiv.org/abs/1111.2760>.
- [3] BAIER, C. a KATOEN, J.-P. *Principles of Model Checking*. Leden 2008. 663-687 s. ISBN 978-0-262-02649-9.
- [4] BEHRMANN, G., DAVID, A. a LARSEN, K. A Tutorial on Uppaal. In:. Leden 2004, sv. 3185, s. 200–236. DOI: 10.1007/978-3-540-30080-9_7. ISBN 978-3-540-23068-7.
- [5] BRICS, A. U. a UNIVERSITY, U. *UPPAAL Documentation* [online]. Uživatelský manuál, 1. vyd. Aalborg, Uppsala: BRICS, Department of Information Technology, 2023.
- [6] DAVID, A., DU, D., LARSEN, K., LEGAY, A. a MIKUCIONIS, M. Optimizing Control Strategy Using Statistical Model Checking. In: *NASA Formal Methods*. USA: Springer Publishing Company, 2013, sv. 7871, s. 352–367. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-38088-4_24. ISBN 978-3-642-38087-7. 5th NASA Formal Methods Symposium, NFM 2013 ; Conference date: 14-05-2013 Through 16-05-2013.
- [7] DAVID, A., LARSEN, K., LEGAY, A., MIKUČIONIS, M. a POULSEN, D. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*. Leden 2015, sv. 17. DOI: 10.1007/s10009-014-0361-y.
- [8] DRLÍČKOVÁ, A. *Návrh logaritmických násobiček*. Brno, 2023. [cit. 2024-02-12]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce MRÁZEK, V. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/142678>.
- [9] FROEHLICH, S., GROSSE, D. a DRECHSLER, R. One Method - All Error-Metrics: A Three-Stage Approach for Error-Metric Evaluation in Approximate Computing. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, s. 284–287. DOI: 10.23919/DATE.2019.8715138.
- [10] HAN, J. a ORSHANSKY, M. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. *The IEEE European Test Symposium*. 2013.

- [11] HARTMANNS, A. a HERMANNS, H. In the quantitative automata zoo. *Science of Computer Programming*. 2015, sv. 112, s. 3–23. DOI: <https://doi.org/10.1016/j.scico.2015.08.009>. ISSN 0167-6423. Fundamentals of Software Engineering (selected papers of FSEN 2013). Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0167642315002580>.
- [12] HENKEL, J., LI, H., RAGHUNATHAN, A., TAHORI, M. B., VENKATARAMANI, S. et al. Approximate Computing and the Efficient Machine Learning Expedition. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. New York, NY, USA: Association for Computing Machinery, 2022. ICCAD '22. DOI: 10.1145/3508352.3561105. ISBN 9781450392174. Dostupné z: <https://doi.org/10.1145/3508352.3561105>.
- [13] HRUDA, P. *Využití přibližného počítání v oblasti zpracování obrazu*. Brno, 2020. [cit. 2024-01-19]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce BÍDLO, M. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/129314>.
- [14] JIANG, H., SANTIAGO, F. J. H., MO, H., LIU, L. a HAN, J. Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications. *Proceedings of the IEEE*. 2020, sv. 108, č. 12, s. 2108–2135. DOI: 10.1109/JPROC.2020.3006451.
- [15] KULKARNI, P., GUPTA, P. a ERCEGOVAC, M. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In: *2011 24th International Conference on VLSI Design*. 2011, s. 346–351. DOI: 10.1109/VLSID.2011.51.
- [16] MA, S. a HUAI, J. Approximate computation for big data analytics. *SIGWEB Newslett.* New York, NY, USA: Association for Computing Machinery. únor 2021, sv. 2021, Winter. DOI: 10.1145/3447879.3447883. ISSN 1931-1745. Dostupné z: <https://doi.org/10.1145/3447879.3447883>.
- [17] MAHDIANI, H. R., AHMADI, A., FAKHRAIE, S. M. a LUCAS, C. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2010, sv. 57, č. 4, s. 850–862. DOI: 10.1109/TCSI.2009.2027626.
- [18] MIGUEL, J. S., BADR, M. a JERGER, N. E. Load Value Approximation. In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 2014, s. 127–139. DOI: 10.1109/MICRO.2014.22.
- [19] MITCHELL, J. N. Computer Multiplication and Division Using Binary Logarithms. *IRE Transactions on Electronic Computers*. 1962, EC-11, č. 4, s. 512–517. DOI: 10.1109/TEC.1962.5219391.
- [20] MITTAL, S. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. mar 2016, sv. 48, č. 4. DOI: 10.1145/2893356. ISSN 0360-0300. Dostupné z: <https://doi.org/10.1145/2893356>.
- [21] MRAZEK, V., HRBACEK, R., VASICEK, Z. a SEKANINA, L. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of

- Approximation Methods. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, s. 258–261. DOI: 10.23919/DATE.2017.7926993.
- [22] POWERS, D. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Mach. Learn. Technol.* Leden 2008, sv. 2.
 - [23] SHALF, J. The future of computing beyond Moore's Law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. Leden 2020, sv. 378, s. 20190061. DOI: 10.1098/rsta.2019.0061.
 - [24] STRNADEL, J. Statistical Model Checking of Approximate Circuits: Challenges and Opportunities. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, s. 1574–1577. DOI: 10.23919/DATE48585.2020.9116207.
 - [25] WAGGENER, W. *Pulse Code Modulation Techniques*. Springer, 1994. 206 s. ISBN 978-0-442-01436-0.
 - [26] WALLACE, C. S. A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*. 1964, EC-13, č. 1, s. 14–17. DOI: 10.1109/PGEC.1964.263830.
 - [27] WU, Y., CHEN, C., XIAO, W., WANG, X., WEN, C. et al. *A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits*. 2023.
 - [28] YESIL, S., AKTURK, I. a KARPUZCU, U. R. Toward Dynamic Precision Scaling. *IEEE Micro*. 2018, sv. 38, č. 4, s. 30–39. DOI: 10.1109/MM.2018.043191123.
 - [29] ZULIANI, P. Statistical model checking for biological applications. Berlin, Heidelberg: Springer-Verlag. srpen 2015, sv. 17, č. 4, s. 527–536. DOI: 10.1007/s10009-014-0343-0. ISSN 1433-2779. Dostupné z: <https://doi.org/10.1007/s10009-014-0343-0>.
 - [30] ŠPANĚL, M. a BERAN, V. *Obrazové segmentační techniky*. Studijní dokumentace. Brno: Vysoké učení technické, Fakulta informačních technologií, 2005 [cit. 2024-01-19]. Dostupné z: http://www.fit.vutbr.cz/~spanel/segmentace/#_Toc125769325.

Příloha A

Obsah paměťového média

/.....	Médium
aux/.....	Pomocné soubory
density/.....	Odhad a generování pst. rozdělení
in/.....	Převzaté zdrojové soubory algoritmu
out/.....	Grafy pravděpodobnostních rozdělení
src/.....	Skripty pro odhad a generování pst. rozdělení
scalability/	Zkoumání škálovatelnosti
models/.....	Převod modelů z Verilogu do UPPAAL
out/.....	Výstupní XML soubory – systémy modelů programu UPPAAL
templates/	Šablony jednotlivých modelů
verilog/.....	Vstupní soubory – modely násobiček ve Verilogu
parse.py.....	Skript pro překlad modelů z Verilogu do UPPAAL
results/	Zpracování výsledků simulací
out_csv/.....	Zpracované výsledky ve formátu csv
pickles/	Pandas DF uložené v souboru pkl obsahující výsledky simulací
plots/	Grafy výsledků simulací
sim_results/.....	Výsledky simulací ve formátu csv
plot_results.py.....	Skript pro vytvoření grafů
print_results.py.....	Skript pro vytvoření výstupních csv souborů
process_results.py ..	Skript pro vytvoření Pandas DF z csv výsledků simulací
thesis/	Zdrojové soubory textu

Příloha B

Návod na zprovoznění

V této příloze je popsáno, jak zprovoznit program UPPAAL, jak pracovat se skriptem pro překlad modelů a jak spouštět simulace v prostředí UPPAAL.

Nástroj UPPAAL

Pro provádění simulací je zapotřebí nainstalovat simulační nástroj UPPAAL. V době vzniku této bakalářské práce byla aktuální verze 5.0. Program lze stáhnout na adrese <https://uppaal.org/downloads/> v závislosti na operačním systému. Dále je nutné se zaregistrovat na adrese <https://uppaal.veriaal.dk/academic.html> pro obdržení licence k používání programu. Pro akademické pracovníky a studenty je zdarma.

Skript parse.py

Pro spouštění skriptu je potřeba prostředí `python3`. Ve skriptu jsou použity pouze standardní moduly `re` a `argparse`.

Skript lze spouštět v příkazové řádce ve tvaru

```
python3 parse.py input_file [-d DISTRIBUTION] [--noout].
```

Argumenty skriptu mají následující význam:

- `input_file` je cesta ke vstupnímu souboru s modelem násobičky napsaným ve Verilogu,
- `-d DISTRIBUTION` je nepovinný argument sloužící pro specifikaci vybraného pravděpodobnostního rozdělení pro generování vstupů. `DISTRIBUTION` může být jeden z názvů rozdělení popsaných v části 4.3,
- `--noout` je nepovinný přepínač sloužící k případnému zakázání vytváření výstupního XML souboru. Vhodné pouze k ladění skriptu.

Výstupní soubory se v aktuální podobě skriptu ukládají do složky `./out`, která má uvnitř téměř stejnou strukturu, jako složka `./verilog` obsahující vstupní soubory.

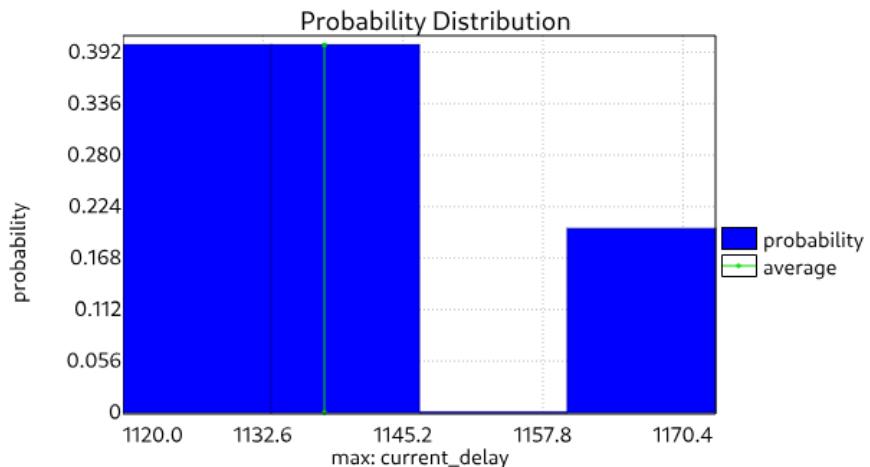
Simulace

Soubor vygenerovaný skriptem `parse.py` je možné otevřít v programu UPPAAL. V části programu nazvané **Verifier** jsou připraveny simulační dotazy. Jednotlivé dotazy lze spouštět tlačítkem **Check**. Po dokončení simulace lze na dotaz kliknout pravým tlačítkem myši a poté kliknout na položku **Simulations (1)** pro otevření vizualizace výsledků dané simulace. Odtud lze exportovat data ve formátu csv. Simulace posledního dotazu může trvat až několik desítek minut či jednotek hodin v závislosti na výkonu procesoru, velikosti operační paměti a také na počtu logických hradel zkoumané násobičky.

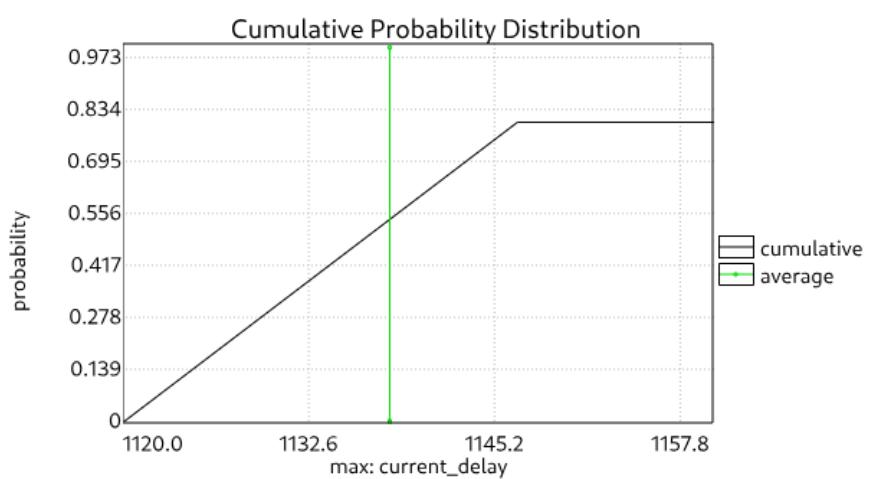
Příloha C

Pravděpodobnostní grafy při odhadu max. nebo min. hodnoty

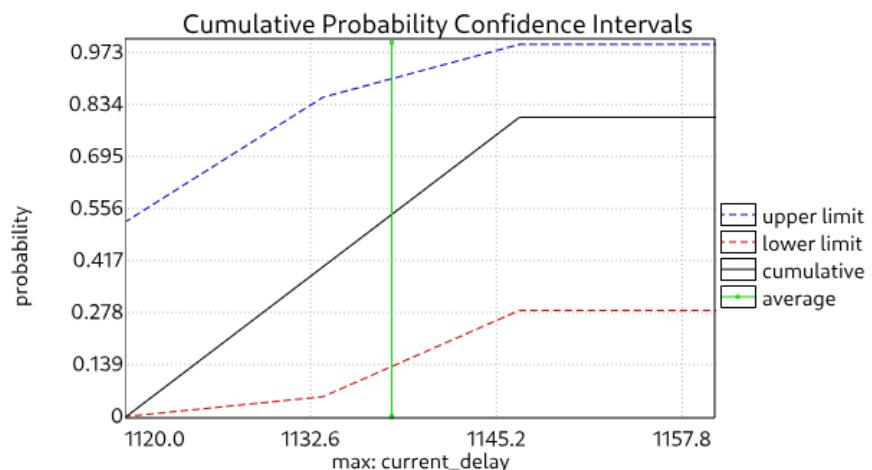
V této příloze jsou uvedeny příklady grafů popsaných v části 3.4.5.



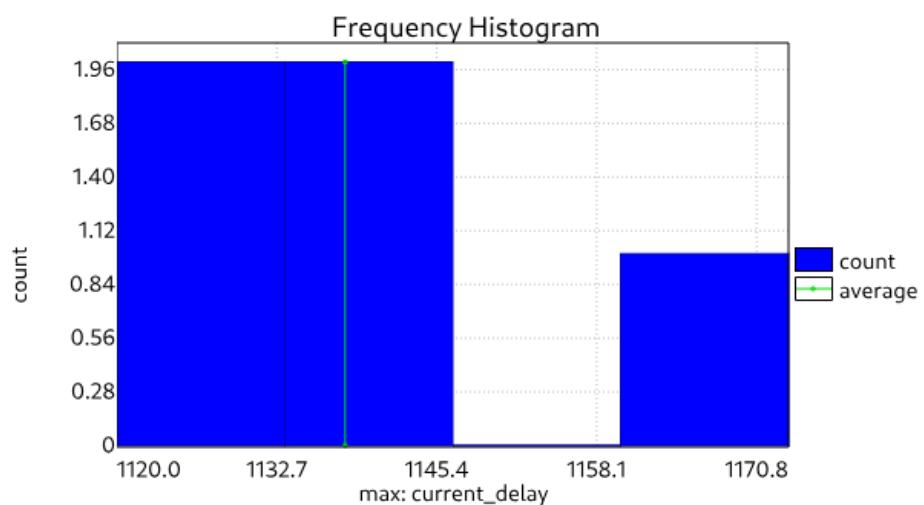
Obrázek C.1: Příklad grafu rozdělení pravděpodobnosti



Obrázek C.2: Příklad grafu rozdělení distribuční pravděpodobnosti



Obrázek C.3: Příklad grafu konfidenčních intervalů distribuční funkce



Obrázek C.4: Příklad histogramu frekvencí výskytu jednotlivých hodnot při simulačních bězích

Příloha D

Pravděpodobnostní rozdělení při generování vstupů korespondující se zvolenými aplikacemi

Výchozím a referenčním rozdělením bylo zvoleno rozdělení nazvané **uni_uni**. Oba vstupy jsou v tomto případě generovány s rovnoměrným rozdělením v intervalu 0 až 255 (minimální a maximální hodnoty, kterých může nabývat 8bitový bezznaménkový integer). Jinými slovy všechny vstupní kombinace mají stejnou pravděpodobnost výskytu.

V této sekci jsou postupně představeny jednotlivé algoritmy a získaná rozdělení. U každého algoritmu se nachází obrázek s grafy naměřených rozdělení. Struktura těchto obrázků je následující:

V grafech (a) a (b) je vidět hustota rozdělení pravděpodobnosti (zkr. PDF z angl. *probability density function*) získaná z reálných naměřených dat z algoritmu. V grafu (a) je zobrazena pravděpodobnost výskytu pro oba vstupy zvlášť, v grafu (b) je to stejné v podobě teplotní mapy. V grafech (c) a (d) jsou zobrazena výsledná odhadnutá pravděpodobnostní rozdělení pro oba vstupy.

Dále je u každého algoritmu vypsán pseudokód generování čísel dle zvolených rozdělení. Součástí kódu ve výsledném souboru pro nástroj UPPAAL je mj. i ošetření překročení hranic (čísla menší než 0 se nastaví na 0, čísla větší než 255 se nastaví na 255), které v pseudokódech není zahrnuto.

Algoritmus Ellipse Mid-Point

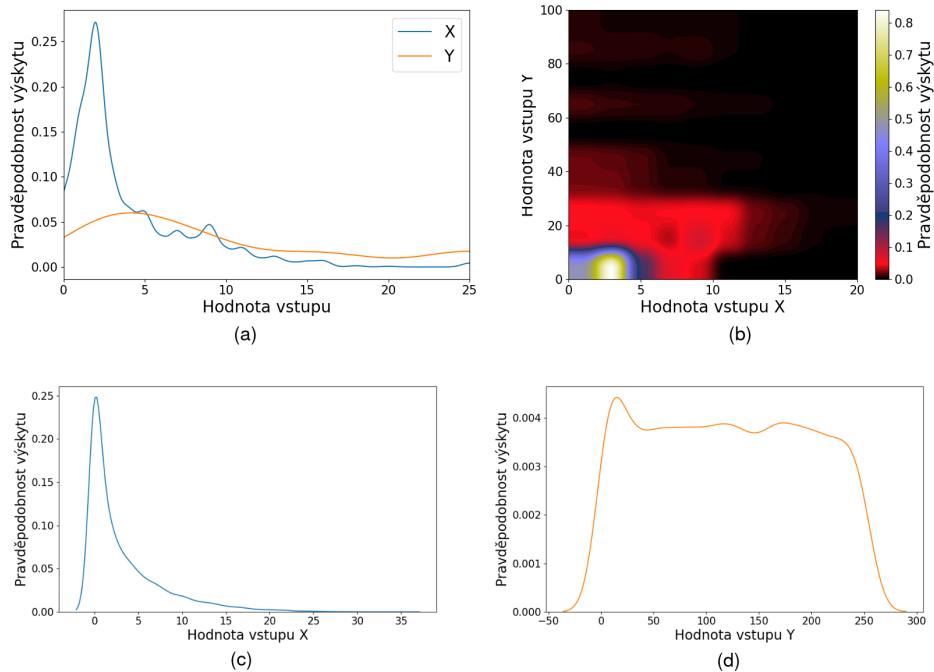
Ellipse Mid-Point je algoritmus používaný v grafice pro rasterizaci elipsy. Na obrázku D.1 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```

1 input_x = int(random_beta(0.5, 5.0) * 40)
2
3 input_y = int(random_uniform(0, 266))
4 if input_y > 255:
5     input_y = int(random_uniform(0, 15))
6
7 if input_y < 2:
8     input_y += 2

```

Pro generování vstupu X bylo zvoleno rozdělení beta s parametry $\alpha = 0,5$ a $\beta = 5$ následně vynásobené hodnotou 40. Vstup Y je generován mírně deformovaným rovnoměrným rozdělením od 0 do 255. V rámci experimentů byla tato dvojice rozdělení označena jako **beta_uni**.



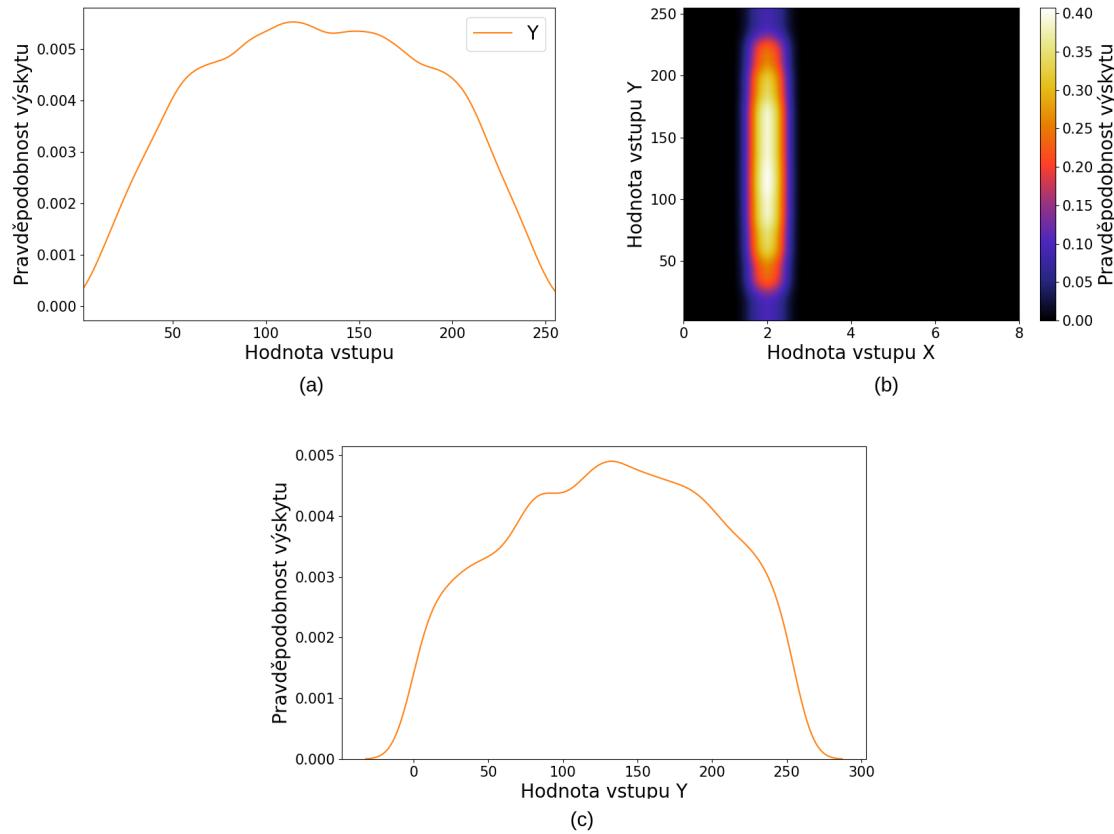
Obrázek D.1: (a) a (b) PDF násobených dvojic rasterizačního algoritmu Ellipse Mid-Point, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů

Bresenhamův algoritmus

Bresenhamův algoritmus je algoritmus používaný v grafice pro rasterizaci úsečky. Na obrázku D.2 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```
1 input_x = 2
2
3 input_y = int(random_normal(150, 30))
4 if input_y < 0:
5     input_y = int(random(100, 255))
```

V algoritmu dochází pouze k násobení různých hodnot číslem 2. Vstupem X je tedy konstanta 2, vstup Y přibližně odpovídá normálnímu rozdělení se střední hodnotou 150 a rozptylem 30. V rámci experimentů byl tento způsob generování čísel označen jako **const_norm**.



Obrázek D.2: (a) a (b) PDF násobených dvojic Bresenhamova rasterizačního algoritmu, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů

Pritchardovo síto

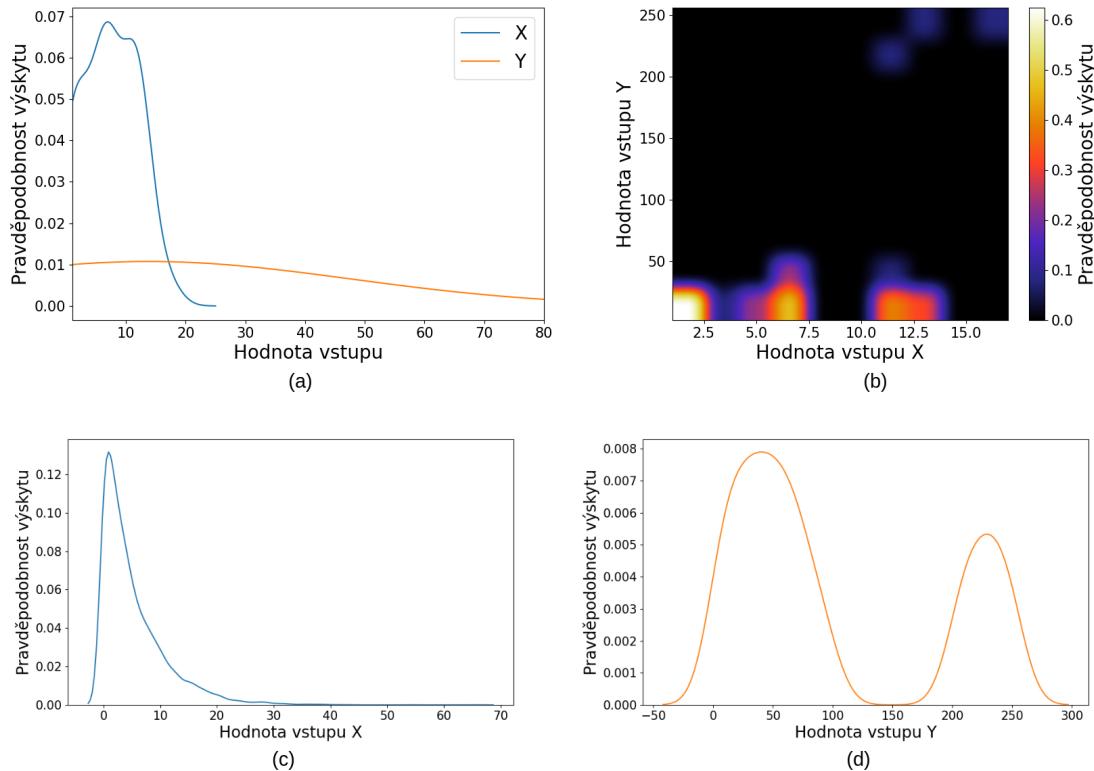
Pritchardovo síto (angl. *Sieve of Pritchard*) je algoritmus z oblasti teorie čísel používaný pro nalezení všech prvočísel menších než je daná hranice. Na obrázku D.3 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```

1 input_x = int(random_gamma(1.0, 0.8) * 7)
2
3 input_y = int(random_uniform(0, 256))
4 if 100 < input_y < 175:
5     input_y = int(random_uniform(0, 75))
6
7 if 175 <= input_y < 200:
8     input_y = int(random_uniform(200, 255))

```

Pro generování vstupu X bylo zvoleno rozdělení gamma s parametry $k = 1.0$ a $\theta = 0.8$ následně vynásobené hodnotou 7. Vstup Y byl generován kombinací několika rovnoměrných rozdělení, což dalo vzniknout dvěma přibližným normálním rozdělením okolo středů 50 a 220. V rámci experimentů byl tento způsob generování čísel označen jako **gamma_2norm**.



Obrázek D.3: (a) a (b) PDF násobených dvojic v algoritmu Pritchardovo síto, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů

Algoritmus Integer Square Root

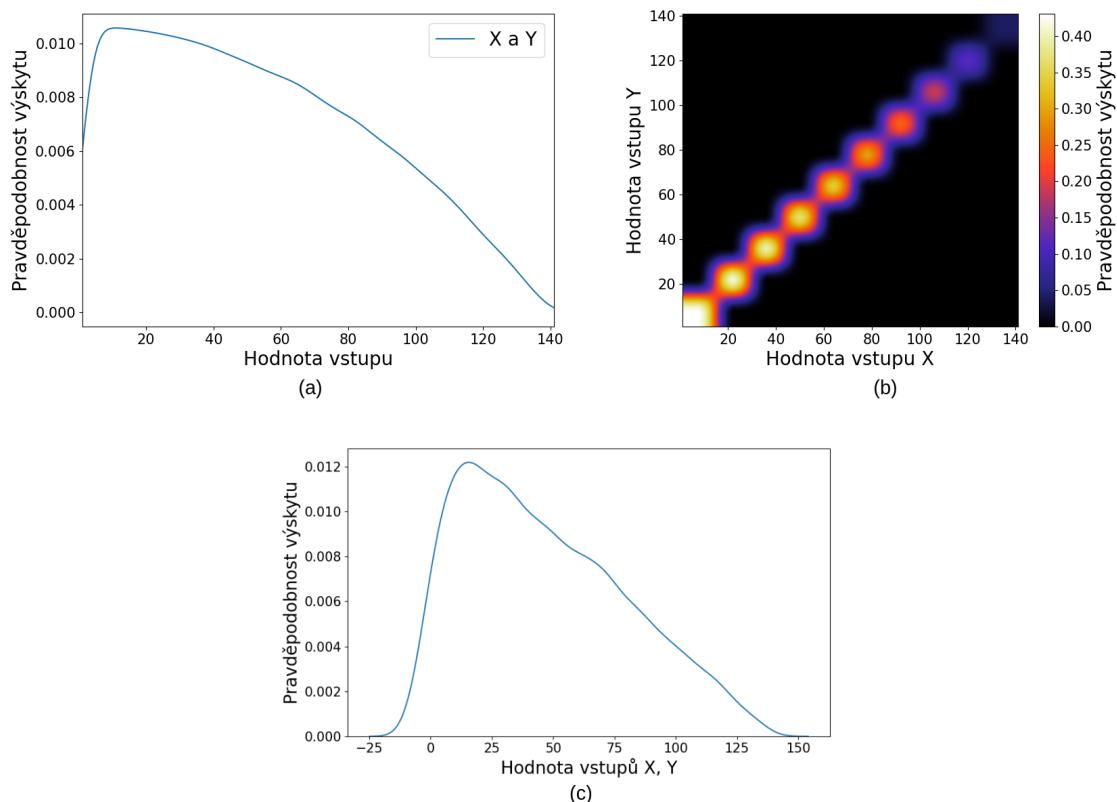
Algoritmus používaný k určení celočíselné odmocniny nějakého čísla, tedy největšího přirozeného čísla, které je menší nebo rovno odmocnině daného čísla. Na obrázku D.4 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```

1 input_x = int(random_triangular(-10, 255, 10))
2
3 input_y = input_x

```

V algoritmu dochází k umocňování na druhou, oba vstupy jsou tedy stejná čísla. Frekventovanější jsou menší hodnoty, proto bylo pro generování zvoleno triangulární rozdělení s minimem v -10, maximem v 255 a modelem v hodnotě 10. V rámci experimentů byl tento způsob generování čísel označen jako **same_triang**.



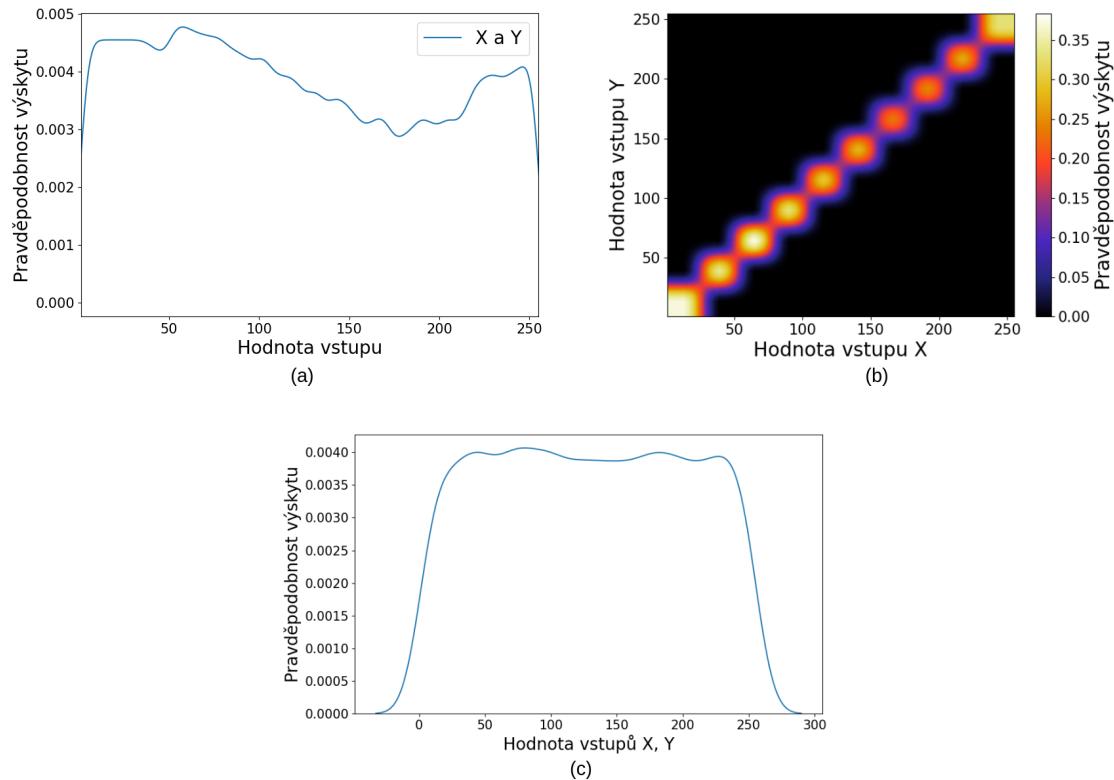
Obrázek D.4: (a) a (b) PDF násobených dvojic v algoritmu výpočtu celočíselné odmocniny, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů

Algoritmus Circle Point to Point

Algoritmus používaný v grafice k rasterizaci kružnice. Na obrázku D.5 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```
1 input_x = int(random_uniform(0, 255))
2
3 input_y = input_x
```

V algoritmu Point to Point dochází, podobně jako u předchozího algoritmu, k umocňování čísel na druhou. V tomto případě ovšem vstupy následují přibližně rovnoměrné rozdělení od 0 do 255. V rámci experimentů byl tento způsob generování čísel označen jako **same_uni**.



Obrázek D.5: (a) a (b) PDF násobených dvojic v algoritmu Circle Point to Point, (c) Pravděpodobnostní rozdělení použité při generování náhodných vstupů

Algoritmus AKS

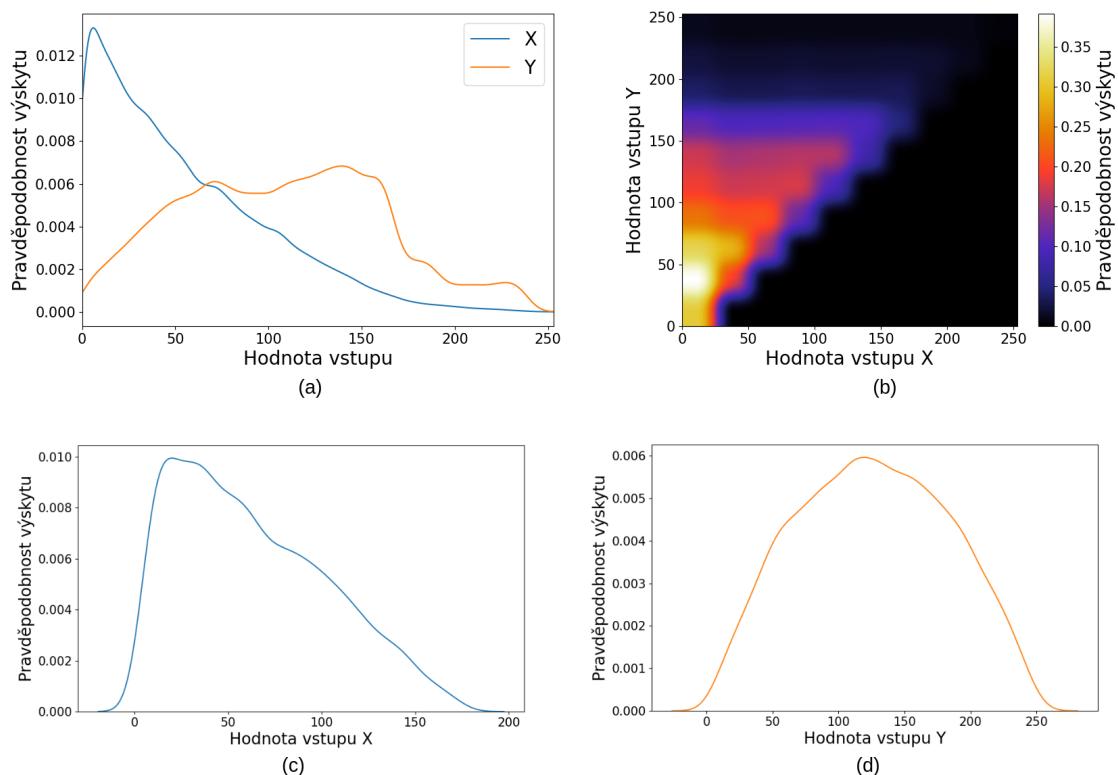
AKS je algoritmus z oblasti teorie čísel sloužící k ověření toho, zda je nějaké dané číslo prvočíslém. Na obrázku D.6 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

```

1 input_x = int(random_triangular(-50, 450, 70))
2 if input_x > 255:
3     input_x = int(random_uniform(0, 150))
4
5 input_y = fint(random_beta(2.0, 2.0) * 255);

```

Rozdělení vstupu X přibližně odpovídá triangulárnímu rozdělení s minimem -50, maximem 450 a modelem v hodnotě 70. Vstup Y byl generován podle rozdělení beta s parametry $\alpha = 2$ a $\beta = 2$. Vygenerovaná hodnota byla následně vynásobena 255 (beta rozdělení figuruje na intervalu 0 až 1). V rámci experimentů byl tento způsob generování čísel označen jako **triang_beta**.



Obrázek D.6: (a) a (b) PDF násobených dvojic v algoritmu AKS, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů

Algoritmus ElGamal

Algoritmus ElGamal slouží k asymetrickému šifrování klíčů. Na obrázku D.7 jsou znázorněna pravděpodobnostní rozdělení vstupů násobičky. Pseudokód zvolených rozdělení je následující:

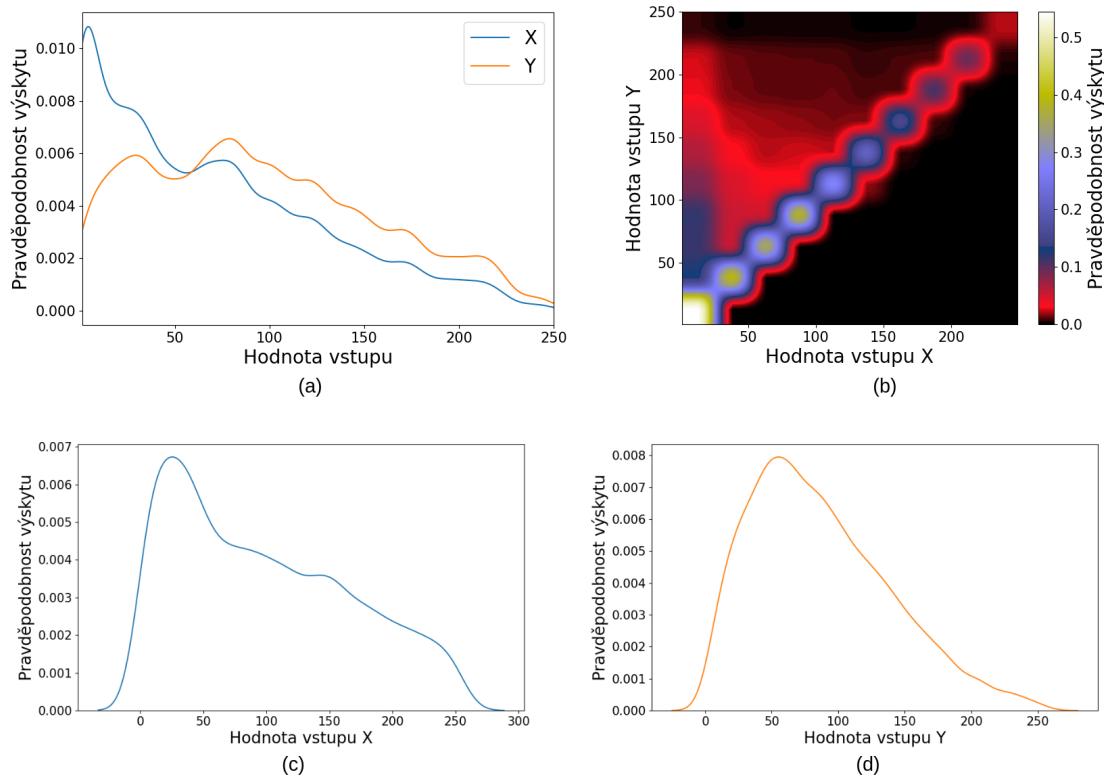
```

1 input_x = int(random_triangular(0, 350, 0))
2 if input_x > 255:
3     input_x = int(random_uniform(0, 50))
4
5 input_y = int(random_weibull(1.7, 1.7) * 60)
6 if input_y > 255:
7     input_y = int(random_uniform(0, 25))

```

Vstup X je generován dle lehce deformovaného triangulárního rozdělení s minimem a modelem 0 a maximem 350. Hodnoty větší než 350 jsou ovšem zahozeny a je místo nich vygenerováno číslo z intervalu 0 až 50, čímž dochází ke zmíněné deformaci.

Vstup Y je generován pomocí Weibulova rozdělení s parametry $k = 1,7$ a $\lambda = 1,7$ deformovaného podobným způsobem jako vstup X. V rámci experimentů byl tento způsob generování čísel označen jako **triang_weibull**.



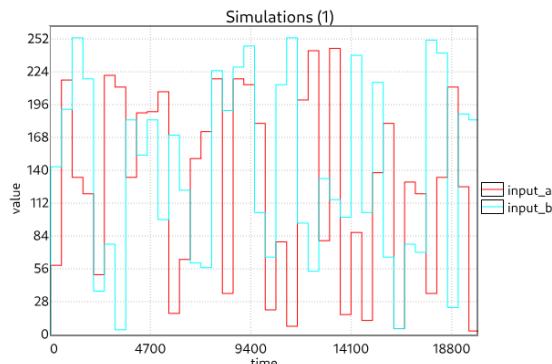
Obrázek D.7: (a) a (b) PDF násobených dvojic v algoritmu ElGamal, (c) a (d) Pravděpodobnostní rozdělení použitá při generování náhodných vstupů

Příloha E

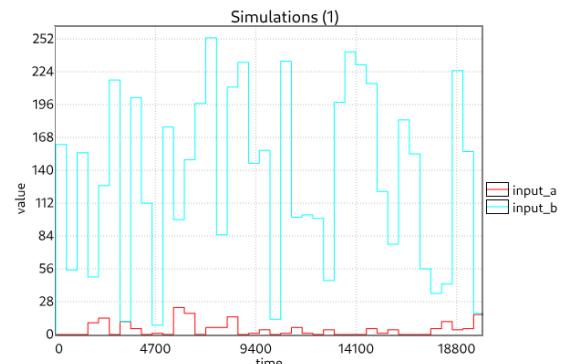
Výsledky ověřovacích simulačních dotazů

Výstupy kontrolních dotazů popsaných v sekci 4.4.

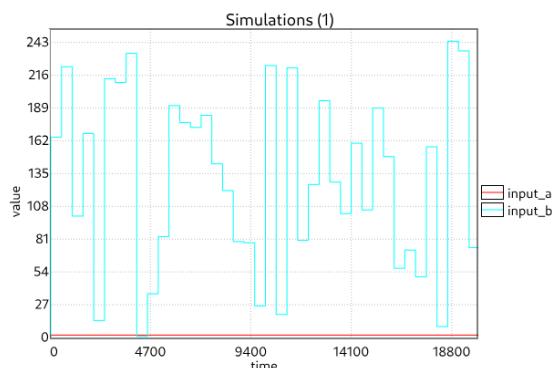
Kontrola vstupních hodnot



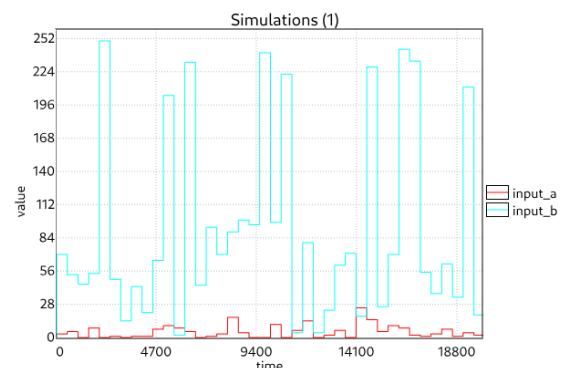
Obrázek E.1: Rozdělení **uni_uni**



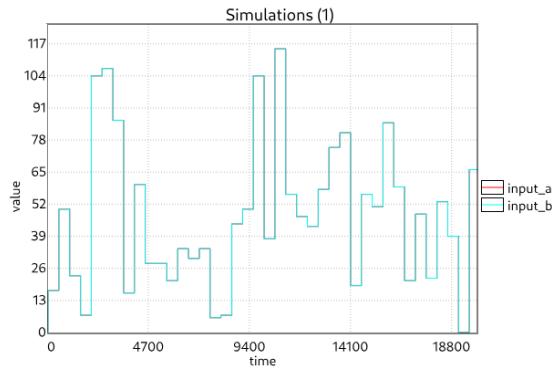
Obrázek E.2: Rozdělení **beta_uni**



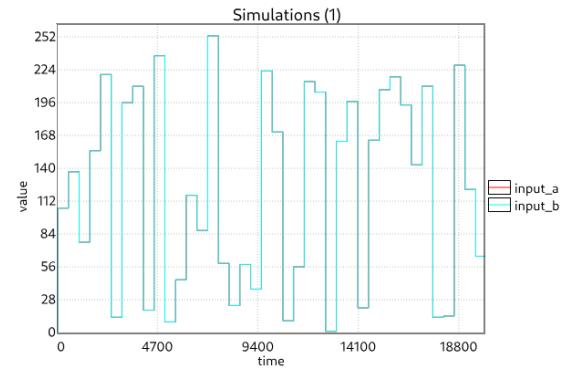
Obrázek E.3: Rozdělení **const_norm**



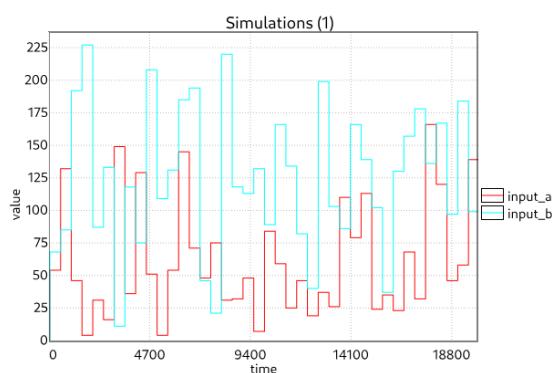
Obrázek E.4: Rozdělení **gamma_2norm**



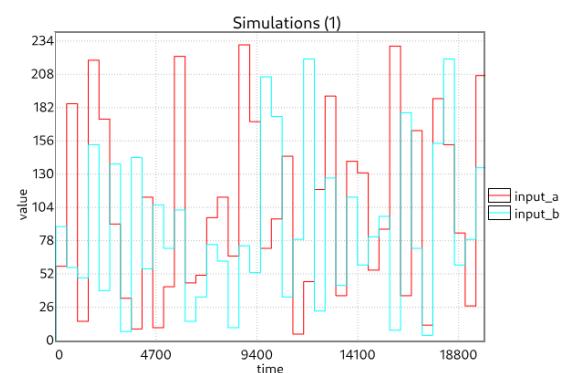
Obrázek E.5: Rozdělení **same_triang**



Obrázek E.6: Rozdělení **same_uni**

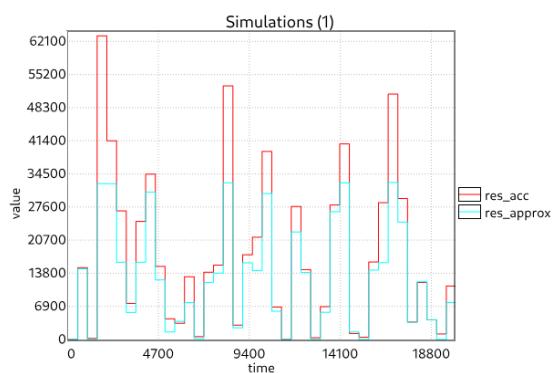


Obrázek E.7: Rozdělení **triang_beta**

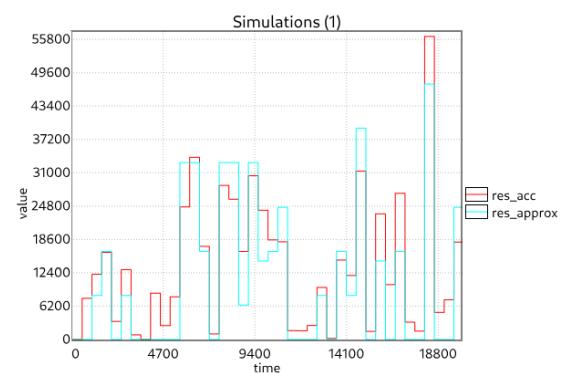


Obrázek E.8: Rozdělení **triang_weibull**

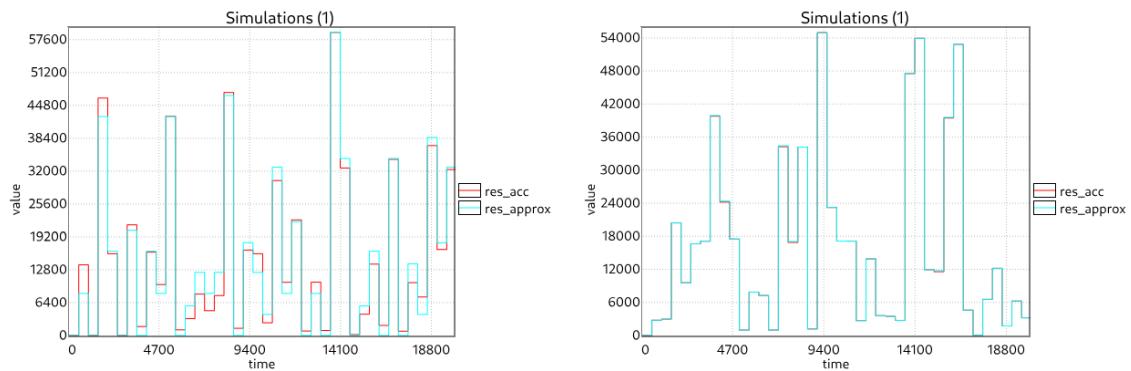
Porovnání přesných a přibližných výstupů



Obrázek E.9: Násobička **mul8u_R36**



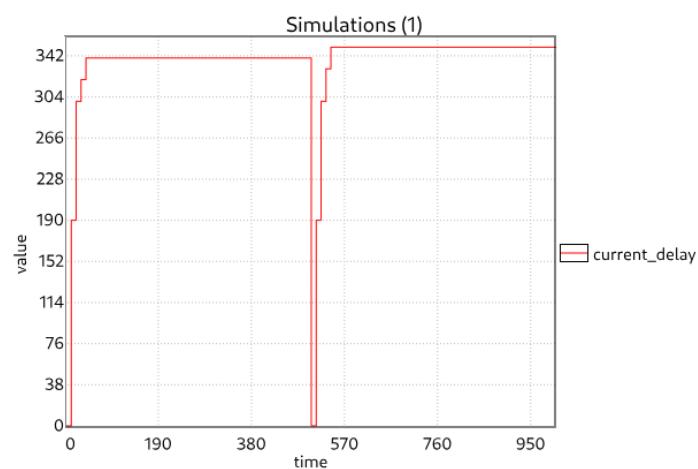
Obrázek E.10: Násobička **mul8u_17MN**



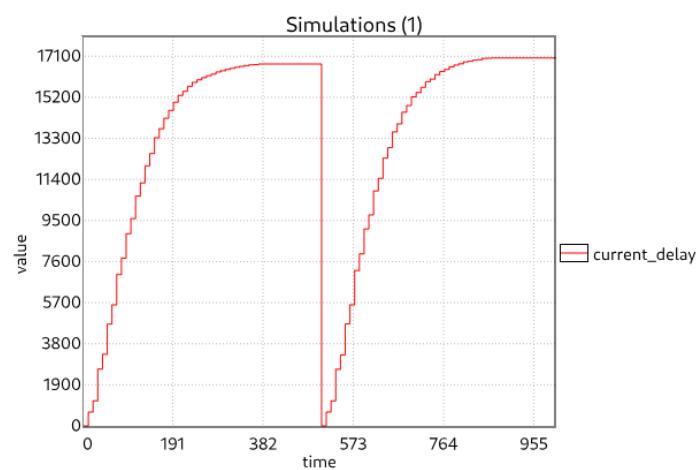
Obrázek E.11: Násobička **mul8u_1A0M**

Obrázek E.12: Násobička **mul8u_12KA**

Sledování zpoždění



Obrázek E.13: Zpoždění násobičky **mul8u_R36**



Obrázek E.14: Zpoždění násobičky **mul8u_12KA**

Příloha F

Podrobné výsledky experimentů

Násobička	Plocha	% pokrytí	Pravděpodobnost chyb	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	14.12	0.97	2342.61	0.59	42880774.66	17726.0	2.94	8.0	115.0	120.0	3.74	9.0
mul8u_17MJ	18.8	14.14	0.89	1406.84	0.53	28646249.92	17030.0	4.76	6.0	105.04	110.0	3.44	8.0
mul8u_R36	60.5	14.13	0.98	3764.64	0.31	36718512.06	32289.0	1.0	10.0	338.56	380.0	12.76	30.0
mul8u_Z9D	220.6	14.09	0.89	3141.56	0.16	32475721.88	31752.0	0.66	13.0	1290.61	1410.0	48.12	91.0
mul8u_17R6	228.5	14.1	0.98	185.57	0.12	306809.19	1905.0	2.56	11.0	5162.93	6320.0	56.0	189.0
mul8u_2NDH	347.8	14.15	0.99	93.52	0.14	169105.55	2709.0	208.0	14.0	4263.88	4890.0	68.36	179.0
mul8u_19T3	395.6	14.09	0.98	63.92	0.04	20974.73	424.0	1.0	14.0	9954.0	10650.0	124.19	312.0
mul8u_NLX	511.5	14.08	0.95	6.06	0.02	2751.55	161.0	1.29	14.0	12874.74	14430.0	163.53	421.0
mul8u_GTR	550.5	14.09	0.84	20.1	0.01	1528.88	125.0	1.0	13.0	13978.72	15470.0	176.54	459.0
mul8u_BGI	561.8	14.13	0.49	289.01	0.03	393955.35	2932.0	0.53	13.0	19263.54	21750.0	184.4	555.0
mul8u_R92	604.5	14.1	0.83	2.84	0.01	212.73	39.0	2.56	13.0	15563.89	17350.0	204.95	498.0
mul8u_ZB3	682.8	14.15	0.07	0.14	0.0	0.27	2.0	0.22	10.0	17238.84	19040.0	230.85	538.0
mul8u_12KA	683.3	14.13	0.09	11.91	0.0	1761.59	192.0	0.28	9.0	16654.07	18160.0	223.03	511.0

Tabulka F.1: Výsledky při generování vstupů s rozdělením uni_uni

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná kvadratická chyba	Průměrná relativní chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	5.74	0.53	403.82	0.63	642255.99	7719.0	1.0	4.0	114.97	120.0	0.0	0.0
mul8u_17MJ	18.8	5.7	0.32	386.49	0.62	597195.3	7470.0	2.02	2.0	104.94	110.0	0.0	3.0
mul8u_R36	60.5	4.42	0.43	177.72	0.34	138906.19	3587.0	1.0	7.0	338.52	380.0	2.88	15.0
mul8u_Z9D	220.6	5.84	0.24	46.86	0.04	25044.73	2544.0	0.46	11.0	1290.63	1440.0	20.32	73.0
mul8u_17R6	228.5	5.69	0.6	112.54	0.47	63296.39	1058.0	1.02	8.0	5163.77	6350.0	3.72	58.0
mul8u_2NDH	347.8	5.74	0.63	42.81	2.33	17614.22	1286.0	208.0	12.0	4264.76	4940.0	8.54	60.0
mul8u_197B	395.6	5.75	0.62	55.24	0.28	9216.48	393.0	1.02	11.0	9952.51	10660.0	18.82	144.0
mul8u_NLX	511.5	5.79	0.59	19.07	0.14	1208.7	155.0	1.0	10.0	12872.55	14360.0	28.13	185.0
mul8u_GTR	550.5	5.77	0.57	16.57	0.14	951.67	125.0	1.0	10.0	13980.68	15520.0	32.09	224.0
mul8u_BG1	561.8	5.8	0.2	46.56	0.04	27770.72	1780.0	0.53	10.0	19247.5	21830.0	36.47	186.0
mul8u_R92	604.5	5.75	0.54	2.71	0.07	139.37	39.0	3.0	10.0	15571.76	17230.0	45.5	269.0
mul8u_ZDF	651.9	5.7	0.15	0.55	0.0	16.31	34.0	0.37	10.0	16358.54	18150.0	50.62	272.0
mul8u_ZB3	682.8	5.77	0.03	0.07	0.0	0.13	2.0	0.22	9.0	17241.52	19190.0	55.97	294.0
mul8u_12KA	683.3	5.8	0.05	6.08	0.01	902.97	192.0	0.29	4.0	16663.53	18130.0	56.81	283.0

Tabulka F.2: Výsledky při generování vstupů s rozdelením beta_uni

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná kvadratická chyba	Průměrná relativní chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	0.39	0.69	117.34	0.90	19667.2	508.0	1.0	3.0	114.99	120.0	0.0	0.0
mul8u_17MJ	18.8	0.39	0.34	115.63	0.90	19121.98	508.0	1.0	1.0	104.95	110.0	0.0	0.0
mul8u_R36	60.5	0.38	0.75	62.46	0.71	5239.86	126.0	1.0	2.0	338.21	380.0	1.43	4.0
mul8u_Z9D	220.6	0.39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1290.57	1410.0	13.65	32.0
mul8u_17R6	228.5	0.39	0.94	116.76	0.99	19667.46	508.0	1.0	5.0	5168.19	6390.0	0.0	0.0
mul8u_2NDH	347.8	0.39	0.99	33.17	0.73	5188.43	126.0	1.0	6.0	4264.8	4940.0	2.97	10.0
mul8u_197B	395.6	0.39	0.99	80.78	0.82	7715.55	182.0	1.0	7.0	9952.15	10650.0	4.05	20.0
mul8u_NLX	511.5	0.39	0.93	24.41	0.36	975.7	62.0	1.0	4.0	12871.78	14340.0	9.97	33.0
mul8u_GTR	550.5	0.39	0.93	30.25	0.39	1254.22	62.0	1.0	4.0	13983.24	15550.0	11.81	37.0
mul8u_BG1	561.8	0.39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	19247.02	21780.0	23.51	58.0
mul8u_R92	604.5	0.39	0.87	14.77	0.21	302.78	30.0	1.0	3.0	15564.57	17340.0	17.85	49.0
mul8u_ZDF	651.9	0.39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16359.14	18420.0	26.08	65.0
mul8u_ZB3	682.8	0.39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	17234.96	19160.0	27.95	69.0
mul8u_12KA	683.3	0.39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16672.32	18400.0	28.33	70.0

Tabulka F.3: Výsledky při generování vstupů s rozdelením const_norm

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů
mul8u_17MN	13.1	4.9	0.67	510.49	0.83	1022758.54	10665.0	5.0	115.02	120.0	0.0	0.0
mul8u_17MJ	18.8	4.93	0.41	521.55	0.83	103542.01	8280.0	5.12	4.0	105.01	110.0	0.01
mul8u_R36	60.5	4.01	0.63	249.63	0.55	196629.08	3888.0	1.0	8.0	338.09	380.0	3.21
mul8u_Z9D	220.6	4.91	0.35	74.28	0.06	55009.73	4864.0	0.48	9.0	1290.75	1420.0	24.22
mul8u_17R6	228.5	4.9	0.8	154.84	0.65	79586.26	1185.0	1.02	8.0	5165.0	6350.0	4.63
mul8u_2NDH	347.8	4.91	0.83	71.8	1.77	32365.21	1301.0	208.0	12.0	4264.34	4950.0	11.01
mul8u_197B	395.6	4.94	0.83	70.8	0.42	11724.64	403.0	1.02	11.0	9951.55	10770.0	23.15
mul8u_NLX	511.5	4.97	0.79	25.49	0.24	1741.28	155.0	1.35	10.0	12872.99	14310.0	34.84
mul8u_GTR	550.5	4.97	0.75	21.21	0.21	1276.35	125.0	1.0	10.0	13978.34	15540.0	39.66
mul8u_BG1	561.8	4.96	0.29	70.92	0.06	51509.66	1786.0	0.53	11.0	19254.78	21870.0	44.54
mul8u_R92	604.5	4.95	0.72	3.77	0.11	187.57	39.0	3.0	9.0	15561.95	17390.0	57.73
mul8u_ZDF	651.9	4.92	0.22	0.67	0.01	25.3	34.0	0.37	9.0	16357.23	18200.0	65.03
mul8u_ZB3	682.8	4.97	0.05	0.09	0.0	0.18	2.0	0.22	9.0	17234.22	19250.0	72.47
mul8u_12KA	683.3	5.01	0.06	10.21	0.01	1725.25	192.0	0.29	6.0	16663.23	18240.0	311.0
												301.0

Tabulka F.4: Výsledky při generování vstupů s rozdelením gamma_2norm

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	
mul8u_17MN	13.1	0.39	0.87	995.26	0.72	32130790.77	17533.0	1.01	7.0	114.93	120.0	2.33	
mul8u_17MJ	18.8	0.39	0.83	910.72	0.93	21805715.51	17284.0	5.12	6.0	105.05	110.0	2.64	
mul8u_Z9D	220.6	0.39	0.93	2637.22	0.22	23174748.65	31752.0	0.49	11.0	1290.6	1420.0	43.41	
mul8u_17R6	228.5	0.39	0.89	298.75	0.28	295517.54	1469.0	1.0	9.0	5162.99	6390.0	37.58	
mul8u_2NDH	347.8	0.38	0.96	131.36	0.64	150980.9	2056.0	208.0	12.0	4263.55	4880.0	51.69	
mul8u_197B	395.6	0.39	0.95	80.62	0.14	25775.88	399.0	1.0	12.0	9955.21	10710.0	97.21	
mul8u_NLX	511.5	0.39	0.9	5.89	0.08	4419.01	161.0	1.0	10.0	12862.65	14350.0	129.08	
mul8u_GTR	550.5	0.39	0.84	25.34	0.07	1822.18	105.0	1.0	9.0	13975.3	15560.0	142.56	
mul8u_BG1	561.8	0.39	0.54	231.18	0.03	400613.36	2938.0	0.37	10.0	19263.65	21800.0	147.81	
mul8u_R92	604.5	0.39	0.75	0.63	0.03	153.83	34.0	1.0	9.0	15562.17	17250.0	500.0	
mul8u_ZDF	651.9	0.58	0.32	0.01	105.93	28.0	0.37	13.0	16352.29	18230.0	184.83	503.0	
mul8u_ZB3	682.8	0.39	0.25	0.49	0.0	0.98	2.0	0.22	8.0	17236.71	19340.0	196.22	569.0
mul8u_12KA	683.3	0.39	0.05	4.48	0.0	463.0	192.0	0.01	6.0	16659.01	18230.0	196.87	512.0

Tabulka F.5: Výsledky při generování vstupů s rozdelením same_triang

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	0.39	0.93	1606.1	0.53	48959181.88	17533.0	1.01	7.0	114.98	120.0	3.77	9.0
mul8u_17MJ	18.8	0.39	0.89	2130.75	0.66	34300636.73	17733.0	5.12	6.0	105.06	110.0	3.86	8.0
mul8u_R36	60.5	0.39	0.97	6731.84	0.36	111108008.89	32289.0	1.0	8.0	338.23	380.0	12.61	30.0
mul8u_Z9D	220.6	0.39	0.97	6560.92	0.25	108295128.03	32258.0	0.5	11.0	1290.64	1410.0	48.32	113.0
mul8u_17R6	228.5	0.39	0.93	265.58	0.17	334967.5	1469.0	1.0	9.0	5170.15	6310.0	61.07	217.0
mul8u_2NDH	347.8	0.39	0.98	190.35	0.22	304076.77	2565.0	208.0	12.0	4261.22	4940.0	69.75	186.0
mul8u_197B	395.6	0.39	0.97	68.63	0.09	26932.71	399.0	1.0	12.0	9952.21	10860.0	126.55	344.0
mul8u_NLX	511.5	0.39	0.93	1.75	0.05	4415.26	161.0	1.0	10.0	12861.77	14410.0	165.21	412.0
mul8u_GTR	550.5	0.39	0.85	22.04	0.04	1649.62	105.0	1.0	9.0	13975.75	15690.0	180.46	433.0
mul8u_BG1	561.8	0.39	0.59	332.9	0.02	513636.13	2938.0	0.37	10.0	19266.83	21900.0	188.28	563.0
mul8u_R92	604.5	0.39	0.77	0.09	0.02	158.92	34.0	1.0	9.0	15568.66	17320.0	211.04	509.0
mul8u_ZDF	651.9	0.39	0.6	0.62	0.01	108.98	28.0	0.37	13.0	16353.46	18250.0	225.07	517.0
mul8u_ZB3	682.8	0.39	0.25	0.5	0.0	0.99	2.0	0.22	9.0	17234.77	19150.0	236.42	565.0
mul8u_12KA	683.3	0.39	0.1	12.64	0.0	1881.38	192.0	0.01	6.0	16660.82	18230.0	235.51	536.0

Tabulka F.6: Výsledky při generování vstupů s rozdelením same_uni

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	24.47	0.94	1908.7	0.58	44016175.32	17662.0	2.88	8.0	115.03	120.0	3.4	9.0
mul8u_17MJ	18.8	24.65	0.88	1130.23	0.5	27626212.71	16384.0	4.94	6.0	104.98	110.0	3.2	8.0
mul8u_R36	60.5	24.56	0.94	2846.44	0.26	19135040.46	28280.0	1.0	10.0	338.5	380.0	12.2	30.0
mul8u_Z9D	220.6	24.7	0.86	2193.26	0.13	15285448.93	29480.0	0.49	13.0	1290.58	1420.0	47.03	91.0
mul8u_17R6	228.5	24.65	0.94	181.07	0.08	311500.42	1925.0	2.16	12.0	5171.66	6490.0	52.38	198.0
mul8u_2NDH	347.8	24.63	0.96	81.97	0.6	136136.6	2117.0	208.0	14.0	4263.25	4880.0	64.3	170.0
mul8u_197B	395.6	24.59	0.94	68.81	0.03	20687.88	431.0	1.02	15.0	9953.64	10700.0	117.1	298.0
mul8u_NLX	511.5	24.58	0.91	5.29	0.01	2629.8	161.0	1.0	15.0	12872.46	14500.0	153.74	393.0
mul8u_GTR	550.5	24.62	0.81	19.56	0.01	1490.17	125.0	1.0	12.0	13980.94	15520.0	167.76	429.0
mul8u_BG1	561.8	24.71	0.48	272.05	0.03	390470.61	2938.0	0.52	13.0	19264.64	21830.0	172.46	547.0
mul8u_R92	604.5	24.44	0.8	2.87	0.0	204.07	39.0	1.0	15.0	15565.35	17320.0	193.87	489.0
mul8u_ZDF	651.9	24.55	0.38	0.69	0.0	56.85	34.0	0.18	13.0	16359.27	18250.0	205.36	497.0
mul8u_ZB3	682.8	24.5	0.06	0.12	0.0	0.23	2.0	0.04	10.0	17221.87	19090.0	216.34	569.0
mul8u_12KA	683.3	24.51	0.08	8.23	0.0	97.34	192.0	0.27	10.0	16667.06	18410.0	214.11	491.0

Tabulka F.7: Výsledky při generování vstupů s rozdelením same_beta

Násobička	Plocha	% pokrytí	Pravděpodobnost chyby	Průměrná absolutní chyba	Průměrná relativní chyba	Průměrná kvadratická chyba	Nejhorší absolutní chyba	Nejhorší relativní chyba	Max. Hammingova vzdálenost	Průměrné zpoždění	Nejhorší zpoždění	Průměr překl. bitů	Max. překl. bitů
mul8u_17MN	13.1	24.07	0.96	2714.61	0.78	32971527.69	17759.0	2.94	8.0	114.97	120.0	2.45	9.0
mul8u_17MJ	18.8	24.01	0.85	1594.32	0.74	21622768.14	16384.0	5.12	6.0	105.03	110.0	2.39	8.0
mul8u_R36	60.5	1.81	0.65	2121.24	0.33	11793979.69	27360.0	1.0	9.0	338.35	380.0	6.96	26.0
mul8u_Z9D	220.6	24.04	0.84	1276.01	0.13	6554886.95	27996.0	0.51	12.0	1290.19	1420.0	44.77	91.0
mul8u_17R6	228.5	24.01	0.97	208.34	0.19	293170.32	1867.0	2.56	11.0	5160.37	6390.0	37.88	184.0
mul8u_2NDH	347.8	24.09	0.99	69.75	0.22	90747.81	2709.0	208.0	15.0	4263.95	4940.0	54.05	169.0
mul8u_1973B	395.6	24.08	0.98	82.02	0.07	21982.34	431.0	1.02	15.0	9952.92	10670.0	101.53	303.0
mul8u_NLX	511.5	24.08	0.95	9.55	0.03	2815.48	161.0	1.0	12.0	12871.16	14370.0	137.21	428.0
mul8u_GTR	550.5	24.11	0.84	21.49	0.02	1548.65	125.0	1.0	12.0	13982.51	15600.0	148.85	409.0
mul8u_BG1	561.8	23.95	0.44	192.93	0.03	2861.95.05	2932.0	0.53	13.0	19251.81	21730.0	153.4	502.0
mul8u_R92	604.5	24.05	0.83	3.39	0.01	210.75	39.0	2.56	14.0	15559.05	17250.0	177.41	448.0
mul8u_ZDF	651.9	23.96	0.4	0.6	0.0	60.71	34.0	0.29	14.0	16355.83	18180.0	189.4	542.0
mul8u_ZB3	682.8	24.03	0.06	0.12	0.0	0.24	2.0	0.04	11.0	17219.58	19110.0	201.25	523.0
mul8u_12KA	683.3	24.12	0.05	4.42	0.0	426.44	192.0	0.22	9.0	16665.62	18200.0	197.97	491.0

Tabulka F.8: Výsledky při generování vstupů s rozdelením triang_weibull