České vysoké učení technické v Praze
Fakulta dopravní

Czech Technical University in Prague
Faculty of Transportation Sciences

Ing. Vít Fábera, Ph.D.

Konstrukce konečného automatu pomocí gramatické evoluce
The Construction of FSM using Grammatical Evolution

Summary

Finite State Machine (FSM) is used as a formal behavioral model in hardware and software area. Generally, the FSM is considered as a model of behavior of elements in system analysis. It is a typical trend to automate design on all levels nowadays – as well on the level of the source code or on the level of the finite state machine. Evolutionary techniques are suitable to create (construct) programs or FSMs.

In [13] FSMs are constructed using standard genetic algorithm. The FSMs are represented by transition and output matrices. Complicated tasks were successfully solved if only the special probability mutation operator was used. But the run time increased to tens minutes. An alternative way to the matrix representation is a language description. If the FSM is implemented in software application, it is described by the `switch` command (the description of the FSM is similar in VHLD or Verilog languages in hardware domain). One of the evolution techniques which makes possible to generate a language description is a *grammatical evolution*. A language describing FSM is defined by the context-free grammar. Linear chromosome called *codon* is decoded and it is rewritten to the language description of the FSM. Genetic operators – crossover and mutation – used in algorithm need not be special; they are standard ones used in genetic algorithms. The fitness function evaluates successful passing of training set which contains corresponding input/output sequences pairs. The runtime of the algorithm was shorten to 1 min. 25 sec. against 25 min. 49 sec.

Souhrn

Konečný deterministický automat se dnes využívá jako formální model chování v oblasti hardware i software, v systémových vědách se hovoří o konečném automatu obecně jako o modelu chování prvku systémů. Současná doba je charakteristická snahou automatizovat návrh na všech úrovních, tedy i na úrovni zdrojového kódu programu nebo na úrovni návrhu vlastního konečného automatu. K automatizované tvorbě programů i automatů je možné využít evoluční techniky.

V práci [13] je konstruován konečný automat genetickým algoritmem nad maticovou reprezentací automatu (tabulkou přechodů a výstupů). Složité úlohy byly řešitelné pouze se speciálním pravděpodobnostním operátorem mutace, kdy vzrostl čas běhu algoritmu až na desítky minut. Alternativou k maticovému vyjádření konečného automatu je jazykový popis automatu, kdy je činnost automatu při jeho softwarové implementaci popsána příkazem `switch` (obdobně je popsán automat při popisu hardware v jazyku VHDL či Verilog). Jedna z evolučních technik umožňujících generovat jazykový popis je *gramatická evoluce*. Jazyk, popisující automat, je definován bezkontextovou gramatikou. Lineární chromozom, zvaný *kodon*, je přepisován pomocí gramatiky na jazykový popis automatu. Genetické operátory křížení a mutace, použité v algoritmu, nemusí být speciální – jsou aplikovány ve standardní podobě používané běžně v genetických algoritmech. Fitness funkce vyhodnocuje úspěšnost zpracování trénovací množiny, která obsahuje páry vstupních a výstupních posloupností. U nejsložitější úlohy došlo ke zkrácení běhu algoritmu na 1 min. 25 sec. oproti 25 min. 16 sec. u původního generického algoritmu.

Klíčová slova

konečný automat, genetický algoritmus, gramatická evoluce, kodon, fitness funkce, bezkontextová gramatika, jazykový popis automatu

Keywords

finite state machine, genetic algorithm, grammatical evolution, codon, fitness function, context-free grammar, language description of the FSM

Obsah

1. Úvod

Konečný deterministický automat se využívá prakticky ve všech odvětvích informatiky - uplatnění nachází jak v oblasti hardwarové, tak i v oblasti softwarové. V systémových vědách, kde se často pohybujeme na vyšší úrovni abstrakce než při návrhu konkrétní hardwarové či softwarové aplikace, se používá jako obecný model chování prvku systému.

Konečný deterministický automat jako formální model chování nalezl širší uplatnění nejprve v oblasti hardware – návrh číslicového systému začíná konstrukcí konečného automatu, kdy návrhář tvoří automatový model často podle slovní, neformální specifikace nebo např. podle zadaných časových diagramů. Automatový model je pak součástí specifikací číslicových systémů, popř. se jím popisují pouze dílčí části, např. typicky chování bloku predikce skoku u procesorů typu RISC je v literatuře popsáno konečným automatem.

V softwarové oblasti byl konečný automat využíván hlavně jako model formálních jazyků, konkrétně model lexikálního analyzátoru, který přijímá regulární jazyk. Širšího uplatnění v praxi došel při rozšíření formálních popisů při návrhu softwarových projektů, kdy si stále rostoucí složitost softwaru vynutila používat důsledně formální modely. Např. jeden z diagramů jazyka UML (Unified Modeling Language) je stavový diagram, což je de facto popis chování konečného automatu.

Současná doba je charakteristická snahou automatizovat návrh hardware/software na všech úrovních, tedy i na úrovni zdrojového kódu programu nebo na úrovni návrhu vlastního konečného automatu. V oblasti digitálního hardware můžeme hovořit také o zdrojovém kódu jako o jedné z alternativ popisu vzhledem k používaným jazykům VHDL, Verilog atd.

V historii se poprvé konstrukce automatu evoluční technikou objevila v 60. letech, ještě před vznikem genetických algoritmů. Algoritmus zvaný „evoluční programování“ byl vytvořen L. J. Fogelem [1]. Autor konstruoval konečné automaty pro predikci vstupních písmen evoluční technikou, pouze však s využitím operace, která se později v genetických algoritmech začala nazývat mutace. Některé práce zabývající se evolučními technikami a konečnými automaty se zaměřovaly na tvorbu automatu bez výstupní

funkce pro účely překladačů (lexikální analyzátor) [3], objevují se práce konstruující evolučními technikami celulární automaty [17].

V práci [13] jsem se zaměřil na konstrukci konečných automatů typu Moore a Mealy standardním genetickým algoritmem [2][4][5] s přihlédnutím k hardwarovým aplikacím. Automat byl reprezentován tabulkou (maticí) přechodů a výstupů (tzv. 2D chromozom). Protože standardní genetický algoritmus nevykazoval dobrou konvergenci pro složitější úlohy, byla jednak upravována kriteriální funkce a vytvořen tzv. „*pravděpodobnostní operátor mutace*“, kdy pravděpodobnost výběru jednotlivých genů k mutaci je úměrná skóre. Skóre je vypočítáváno během vyhodnocení kriteriální funkce a kvantifikuje správnost jednotlivých přechodů a výstupů, neboli přechody a výstupy, které se jeví jako špatně vygenerované, mají větší pravděpodobnost mutace. Použitím tohoto operátoru došlo ke zlepšení konvergence genetického algoritmu, ale prodloužil se čas výpočtu. Proto jsem se zaměřil na možnost využití jiné evoluční techniky, konkrétně *gramatické evoluce* a jazykové reprezentace automatu. Navíc, právě výstup algoritmu gramatické evoluce může být přímo základem implementace takového automatu v jazycích VHDL nebo Verilog.

2. Evoluční techniky při konstrukci konečného automatu

2.1 Konečný deterministický automat

V dalším textu budeme uvažovat běžně zavedenou definici konečného deterministického automatu. Konečný deterministický automat [8] [11] je uspořádaná pětice $A = \langle \mathrm{X}, \mathrm{Y}, \mathrm{Q}, \delta, \lambda \rangle$, resp. šestice $A = \langle \mathrm{X}, \mathrm{Y}, \mathrm{Q}, \delta, \lambda, Q_0 \rangle$, kde jednotlivé složky mají následující význam:

X	..	vstupní abeceda (konečná množina všech vstupních písmen, resp. symbolů)
Y	..	výstupní abeceda (konečná množina všech výstupních písmen, resp. symbolů)
Q	..	stavová abeceda (konečná množina všech vnitřních stavů)
δ	..	stavově přechodová funkce (tj. zobrazení $\delta : \mathrm{X} \times \mathrm{Q} \to \mathrm{Q}$, které určuje následný vnitřní stav ze vstupního písmene a současného vnitřního stavu)
λ	..	výstupní funkce (tj. zobrazení $\lambda : \mathrm{X} \times \mathrm{Q} \to \mathrm{Y}$, resp. $\lambda : \mathrm{Q} \to \mathrm{Y}$)
Q_0	..	počáteční vnitřní stav ($Q_0 \in \mathrm{Q}$)

Automat, který má definován počáteční vnitřní stav, se označuje jako *automat iniciální*, na rozdíl od automatu bez definovaného počátečního stavu, který označujeme jako *automat neiniciální*. Z praktického hlediska má význam pouze automat iniciální, protože po digitálním obvodu, programu, obecně po reálném technickém zařízení zcela samozřejmě požadujeme, aby byl při uvádění do činnosti (např. zapnutí napájecího napětí, spuštění software) v definovaném stavu.

Podle tvaru výstupní funkce se rozlišují dva typy automatů. Závisí-li výstupní písmeno pouze na aktuálním vnitřním stavu

($\lambda : Q \rightarrow Y$), hovoří se o *Mooreově* automatu, pokud je výstupní písmeno určeno aktuálním vstupním písmenem a aktuálním vnitřním stavem ($\lambda : X \times Q \rightarrow Y$), jedná se o *Mealyho* automat. Oba typy automatů jsou ekvivalentní a vzájemně převoditelné. Mooreův automat ekvivalentní určitému automatu typu Mealy má zpravidla více vnitřních stavů, Mealyho automat realizovaný jako digitální hardware mívá složitější výstupní funkci, ale může mít méně přechodů, není to ovšem pravidlem.

Konečný automat popisuje chování sekvenčního systému. Alternativním popisem chování sekvenčního systému je *sekvenční zobrazení* [8]. Sekvenční zobrazení $\eta = \Phi_S(\varphi)$ zobrazuje posloupnost vstupních písmen φ libovolné konečné délky na posloupnost výstupních písmen η konečné délky. Aby mohlo být zobrazení nazváno sekvenčním, musí splňovat dvě vlastnosti [8], [9]:

1. Zachovávat délku posloupnosti: má-li vstupní posloupnost délku *n*, musí mít i výstupní posloupnost délku *n*.
2. Zachovávat počáteční úsek posloupnosti: mají-li dvě vstupní posloupnosti φ_1 a φ_2 stejný počátečný úsek délky *k*, pak výstupní posloupnosti musí mít stejné počáteční úseky alespoň délky *k*.

2.2 Množiny vstupních a výstupních písmen, vnitřní stavy

Vstupní a výstupní písmena jsou v obecném případě symbolická, tedy množina vstupních písmen s kardinalitou $x = |X|$ má podobu $X = \{X_0, X_1, \ldots, X_{x-1}\}$ a množina výstupních písmen s kardinalitou $y = |Y|$ má podobu $Y = \{Y_0, Y_1, \ldots, Y_{y-1}\}$. Při řešení konkrétní úlohy konstrukce konečného automatu jsou známy nejen počty vstupních a výstupních písmen *x*, *y*, ale i samotné množiny vstupních a výstupních písmen. Vstupní a výstupní písmena mají v oblasti digitálního hardware podobu jedno a vícebitových dvojkových čísel. Pro experimenty s konstrukcí automatu pomocí evolučních technik je výhodnější ponechat množiny vstupních písmen symbolické tak, jak je uvedeno výše. Počáteční index písmene vždy uvažujeme o

hodnotě 0. Požadavek je rozumný i z hlediska implementace algoritmů v jazyce C, ve kterém je minimální hodnota indexu polí automaticky rovna 0. Nadále jsou tedy symbolická vstupní a výstupní písmena reprezentována celými čísly. Konkrétně, vstupní písmena $X_0, X_1, \ldots, X_{x-1}$ jsou reprezentována celými čísly $0,1,\ldots,x-1$, výstupní písmena $Y_0, Y_1, \ldots, Y_{y-1}$ jsou reprezentována rozsahem $0,1,\ldots,y-1$. Formálně zapsané převodní funkce mají tvar:

- pro množinu vstupních písmen X je funkce $f_X : \mathrm{X} \rightarrow \mathrm{N}^0$ taková, že $f_X : (X_i) = i$; N^0 je množina přirozených čísel včetně 0.
- pro množinu výstupních písmen Y je funkce $f_Y : \mathrm{Y} \rightarrow \mathrm{N}^0$ taková, že $f_Y : (Y_i) = i$; N^0 je množina přirozených čísel včetně 0.

Stejným způsobem, tedy celými čísly, předpokládáme reprezentaci množiny vnitřních stavů. Označíme-li počet vnitřních stavů automatu $q = |\mathrm{Q}|$, pak množina vnitřních stavů $\mathrm{Q} = \{Q_0, Q_1, \ldots, Q_{q-1}\}$ je reprezentována množinou čísel $0,1,\ldots,q-1$. Převodní funkce je analogická funkcím pro vstupní a výstupní písmena. Na rozdíl od množin vstupních a výstupních písmen, kde jejich počet vyplývá ze zadání úlohy, není kardinalita množiny vnitřních stavů předem známa (není známo, kolik vnitřních stavů bude výsledný konečný automat mít). Problém lze řešit tak, že parametrem algoritmu je omezení maximálního počtu vnitřních stavů, který může automat mít.

Počáteční stav každého automatu je vždy stav Q_0 a není v průběhu evoluce měněn. Podmínka není příliš omezující – proces selekce vybere ty automaty, které jsou evolučně konstruovány od počátečního stavu Q_0.

2.3 Trénovací množina a kriteriální funkce

Zadání úlohy konstrukce automatu evoluční technikou je přirozené definovat jako množinu dvojic vstupních a výstupních posloupností (dále trénovací množinu). Stejný přístup existuje i v obdobných pracích [1], [3] (ve [3] byla dvojice redukována na vstupní posloupnost a třídu koncového stavu). Částečný návod, jak mají dvojice posloupností vypadat, dává definice sekvenčního zobrazení. Posloupnosti musejí především splňovat dvě daná kritéria, tj. být stejně dlouhé a zachovávat počáteční úsek posloupnosti.

V práci [1] měřila kriteriální funkce procentuální úspěšnost predikce symbolů v posloupnosti. Automat přijal vstupní posloupnost symbolů X_0, …, X_{n-1} o délce n. Výstup automatu byl v každém taktu porovnán s následným vstupním symbolem ve vstupní posloupnosti.

Obecnější je, na rozdíl od [1], jestliže trénovací množina obsahuje více dvojic posloupností, nikoliv pouze jedinou dvojici. U každého automatu je vypočítána výstupní posloupnost jako odezva na vstupní posloupnost a porovnána s požadovanou výstupní posloupností. Pro vyhodnocení porovnání jsem navrhl dvě metody [13] [14]. U prvé, striktnější, metody, která se odlišuje od běžných metod hodnocení v genetických algoritmech, je spočítán poměr

$$P = \frac{l}{n}, \tag{1}$$

kde n je délka trénovací posloupnosti a l je délka počátečního úseku výstupní posloupnosti, která se shoduje s požadovanou posloupností. Jinak řečeno, jakmile testovací algoritmus zjistí rozdílný výstupní symbol v generované a požadované výstupní posloupnosti, ukončí testování a zpracovaná délka posloupnosti je hodnota l. První metodu dále nazývám PARTIAL (PART).

Ve druhé metodě je definován poměr celkový, tj. poměr

$$P = \frac{n_0}{n}, \tag{2}$$

kde n je délka trénovací posloupnosti a n_0 je celkový počet shodných výstupních symbolů v generované a požadované výstupní posloupnosti. Druhá metoda je označována jako ALL a shoduje se

s přístupem v [1] a běžnými přístupy v genetických algoritmech (celkový poměr).

Poměr P je vypočítán pro každou dvojici v trénovací množině a jako hodnota kriteriální funkce je proveden „pesimistický odhad“, tj. nejhorší poměr:

$$pom = min\ P_i \text{ pro } i=1,...,m, \tag{3}$$

kde m je počet párů v trénovací množině a P_i je vypočítaný poměr (1) nebo (2) pro i-tý pár v trénovací množině. Pesimisticky definovaná kriteriální funkce zajišťuje, aby byl automat geneticky vyvíjen od počátečního stavu, podobně, jako je tvořen návrhářem.

V teorii automatů je definována *ekvivalence automatů z hlediska chování*. Více o ekvivalenci pojednává literatura [8], [9]. Dva ekvivalentní automaty se mohou lišit i počtem stavů. Proto se po skončení návrhu automatu provádí tzv. *minimalizace počtu vnitřních stavů*. Algoritmus minimalizace je deterministický, je založen na vyhledávání tzv. tříd ekvivalentních stavů pro úplně určené automaty, pro neúplně určené automaty jsou vyhledávány třídy slučitelných stavů [8], [9]. Pro hardwarové aplikace má minimalizace počtu vnitřních stavů význam, protože zmenší počet vnitřních proměnných a tím počet použitých klopných obvodů, zjednoduší se i kombinační část. V práci [3] byla uvažována minimalizace slučováním stavů během geneze automatů, počet stavů automatu ale kriteriální funkce nehodnotí. V práci [1] autor uvažoval počet vnitřních stavů pouze v rámci několika experimentů, velikost automatu přičetl k procentuální úspěšnosti predikce automatu jako penalizaci s váhou 0,01, resp. 0,05.

Jiný možný přístup je zahrnout požadavek na minimální počet vnitřních stavů do kriteriální funkce při každém testování automatu (jedince). Parametrem algoritmu je zadaný počet vnitřních stavů q jako horní mez. Je zřejmé, že graf přechodů mající q vnitřních stavů, nemusí být souvislý. Počáteční stav automatu je vždy Q_0, proto je u každého automatu vypočítána velikost silné komponenty souvislosti k od stavu Q_0, neboli počet vnitřních stavů k, které jsou dosažitelné z počátečního stavu Q_0.

Prvotní je samozřejmě činnost automatu, kritérium minima počtu stavů je až sekundární požadavek. Proto je účelné definovat kriteriální funkci pro úlohu konstrukce automatu evoluční technikou

jako dvojdimenzionální a na množině dvojic definovat *lexikografické* uspořádání.

Tedy, velikost komponenty souvislosti tvoří druhou dimenzi výsledné kriteriální funkce:

$$F = [pom, k] \tag{4}$$

kde *pom* je minimální poměr *min* P_i, viz (3)
k je velikost komponenty souvislosti.

Je definováno lexikografické uspořádání, nutné pro porovnání automatů.

Definice:

Mějme automat *A* s hodnotou kriteriální funkce $F_1 = [pom_1, k_1]$ a automat *B* s hodnotou kriteriální funkce $F_2 = [pom_2, k_2]$. Definujeme uspořádání:

$F_1 < F_2$ *(*A je „horší automat“ než B), právě tehdy, když

$pom_1 < pom_2$ nebo *(*$pom_1 = pom_2$ a $k_1 > k_2$*)* (5)

Neformálně, horší automat je ten, který zvládne úspěšně generovat kratší posloupnost. V případě rovnosti poměru je horší automat s větším počtem vnitřních stavů.

Jeden krok testu znamená simulaci přechodu automatu a porovnávání výstupního písmene. Krok testu má jednotkovou složitost, provede se *v* krát pro jeden automat, kde *v* je celkový počet vstupních písmen v trénovací množině.

Výpočet velikosti komponenty souvislosti příliš neprodlouží čas výpočtu kriteriální funkce, neboť je implementován algoritmem s lineární složitostí $O(q+h)$, kde *q* je maximální počet vnitřních stavů a *h* je počet hran grafu. Počet hran je u každého automatu rozdílný. Za předpokladu, že z každého uzlu (tj. stavu) může vycházet maximální počet hran roven počtu vstupních písmen *x* (některé mohou být multihranami), lze omezit počet hran v grafu horním limitem *qx*.

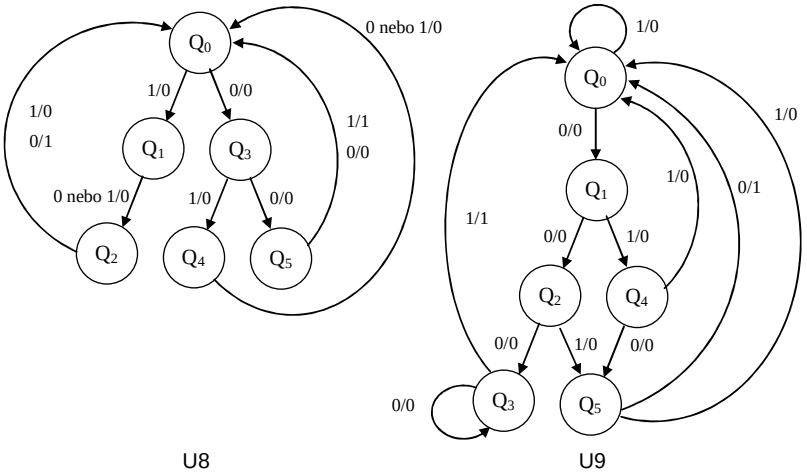
2.4 Automaty generované genetickým algoritmem

V práci [13] byl genetický algoritmus testován na deseti úlohách (tabulka 1), které jsem za tímto účelem vytvořil, pro velikosti generací $N = \{100, 500, 1000, 10\,000\}$ podle složitosti úlohy. Sloupec označený v tabulce „Počet stavů“ znamená počet stavů automatu správného řešení, tj. grafu automatů, který jsem navrhl jako vzorové řešení [13].

Označení úlohy	Charakteristika	Typ automatu	Počet stavů
U1	Fogelova úloha	Mealy	8
U2	Generátor pulsu	Mealy i Moore	3
U3	Rozpoznávání sudého počtu jedniček v sériové posloupnosti	Mealy	3
U4	Rozpoznávání lichého počtu jedniček v sériové posloupnosti	Mealy	3
U5	Řízení doplňování vody do rezervoáru	Moore	2
U6	Čítač mod 8	Moore	8
U7	Čítač mod 8 se vstupem povolujícím čítání	Moore	8
U8	Rozpoznávání tříbitových čísel 1, 3, 4 v sériové posloupnosti dat	Mealy	6
U9	Rozpoznávání čtyřbitové sériové posloupnosti	Mealy	6
U10	Korobovův automat (čítač)	Moore	63

Tabulka 1: Testovací úlohy

Obtížné úlohy pro genetický algoritmus byly úlohy označené U8, U9. Oba grafy přechodů mají jako podgraf strom, obrázek 1.



Obrázek 1: Obtížné úlohy pro genetický algoritmus

Správný automat byl pro obě úlohy nalezen pouze v případě, byl-li algoritmus vhodně parametrizován: ve fitness funkci byla použita metoda PARTIAL a byl aktivován pravděpodobnostní algoritmus mutace a velikost generace 10 000 jedinců. Počet iterací byl omezen na hodnotu 1000. Čas běhu algoritmu pro úlohu U8 byl 4 min. 16 sec, čas běhu algoritmu pro úlohu U9 byl 25 min. 49 s na notebooku HP Compaq nx9030 s procesorem Intel Pentium 1,6 GHz, 512 MB RAM.

Při použití pravděpodobnostního algoritmu mutace se během provedení testu u každého přechodu a výstupu počítá skóre podle úspěšnosti (přičítána a odečítána hodnota 1). Po provedení testu automatu je vypočítána hodnota poměru $pm = \frac{skóre}{n}$ pro každý přechod a výstup, kde n je celkový počet kroků testu (je-li n=0, pak $pm = 0$). Poměry leží v intervalu $\langle -1, 1 \rangle$. Hodnota -1 znamená, že

přechod (výstup) je absolutně úspěšný. Poměr *pm* je transformován pomocí lineárního vztahu:

$$p = a(pm + 1) + d \tag{6}$$

Parametry $a > 0, d \geq 0$ reprezentují škálování a posuv. Byly zvoleny dvě dvojice hodnot $a = 1, d = 0$ a $a = 1, d = 0{,}2$. Hodnoty *p* jsou normalizovány, aby jejich suma počítaná přes celou matici byla rovna hodnotě 1,0. Normalizované hodnoty jsou pravděpodobnostmi mutací jednotlivých přechodů (výstupů). Pravděpodobnost mutace je úměrná „kvalitě“ přechodu (výstupu).

Rozbor ukázal, že u úlohy U9 během evoluce často konverguje řešení k automatu se smyčkami. Je to přirozené, protože výstupní posloupnosti obsahují dlouhé podposloupnosti nul následované jednou jedničkou. Pro tento typ úloh by bylo tedy vhodné „přinutit“ genetický algoritmus, aby se podgraf grafu přechodů podobal stromu, tedy omezit tvorbu smyček. Řešením je přirozeně zahrnout kritérium do fitness funkce.

Fitness funkce byla rozšířena o další dimenzi, a to počet smyček. Upravená fitness funkce byla definována

$$F = [pom, k, l] \tag{7}$$

kde poměr *pom* je obecně počítán metodou ALL nebo PARTIAL, *k* je počet vnitřních stavů a *l* je počet smyček v grafu přechodů. Uspořádání je definováno opět jako lexikografické. Analogicky, nechť $F_1 = [pom_1, k_1, l_1]$ a $F_2 = [pom_2, k_2, l_2]$. Pak $F_1 < F_2$ (A_1 je horší automat než A_2) právě tehdy, když

$pom_1 < pom_2$ nebo ($pom_1 = pom_2$ a $k_1 > k_2$) nebo ($pom_1 = pom_2$ a $k_1 = k_2$ a $l_1 > l_2$)

Nová kriteriální funkce byla opět aplikována na úlohu U9 s parametrizací P4P10000 (50 běhů algoritmu). Správné řešení bylo poprvé nalezeno v průměru v 395. generaci oproti 428. generaci s původní dvojdimenzionální kriteriální funkcí. Průměrná hodnota nejlepších řešení v generaci se bohužel nezvýšila.

Poznámka:

Definici kriteriální funkce je možné rozšířit o další dimenze a hodnotit tak několik dalších parametrů automatu. Např., je-li uvažováno jako další kritérium počet neurčených přechodů a výstupů automatu *np*, je definována kriteriální funkce následovně:

$$F = [pom, k, l, np] .$$

3. Gramatická evoluce

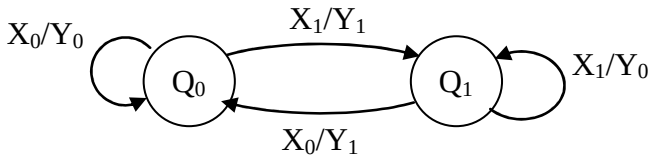
Gramatická evoluce byla poprvé představena autory O'Neilem a Ryanem [18], [19], v české literatuře např. v [6]. Gramatická evoluce slouží ke konstrukci programů evolučním principem, stejně jako genetické programování [6], [7], [10], [16], ale gramatická evoluce vychází ze standardních genetických algoritmů. Gramatická evoluce má některé výhody oproti genetickému programování. Především, jedinci jsou kódováni jako jednodimenzionální pole celých čísel, nazývaná *kodony*, namísto stromů používaných v genetickém programování. Syntaxe jazyka je definována gramatikou, zpravidla bezkontextovou [12], [15]. Kodon je dekódován a transformován na program, který je jedincem reprezentován. Geny kodonu určují, které pravidlo gramatiky má být v daný okamžik aplikováno. Operace křížení a mutace jsou standardní, převzaté ze standardního genetického algoritmu. Aplikace operátorů nemohou produkovat neplatné individuum.

Je-li konečný automat implementován softwarově, je vnitřní stav uložen v proměnné (celočíselného typu, nebo lépe výčtového typu) a činnost automatu je popsána cyklem, v jehož těle je příkaz typu přepínač (`switch` v jazyce C). Větvení je determinováno vnitřními stavy automatu. V každé větvi jsou pak podmínky pro jednotlivá vstupní písmena. Výkonná část podmínky představuje přiřazení nového stavu a výstupního symbolu, resp. provedení nějaké akce v obecném případě. Tedy, konečný automat je možné kromě tabulek (matic) přechodů a výstupů reprezentovat také jazykově. Gramatická evoluce představuje vhodnou alternativu pro evoluční generování jazykového popisu automatu. Gramatická evoluce ve spojení s konstrukcí automatu dosud není často využívanou technikou.

3.1 Využití gramatické evoluce při konstrukci konečného automatu

Softwarová implementace konečného automatu je založena na cyklu, v jehož těle je příkaz typu přepínač, viz odst. 2, kap. 3. Pro ilustraci, konečný automat a jeho odpovídající programová

implementace v pseudokódu odvozeném od jazyka C je na obrázku 2.




```
while(1)
{
  read(input);
  switch (state)
  {
    case Q0: if(input==X0)
             {
                // následný stav a výstup
                state=Q0; output=Y0;
                break;
             }
             if(input==X1)
             {
               state=Q1; output=Y1;
               break;
             }
    case Q1: if(input==X0)
             {
               state=Q0; output=Y1;
               break;
             }
             if(input==X1)
             {
               state=Q1; output=Y0;
               break;
             }
  }
}
```

Obrázek 2: Jazyková reprezentace automatu

Klíčovou částí kódu je samozřejmě příkaz **`switch`** v těle cyklu **`while`**.

Konstruovaný automat má symbolické vnitřní stavy Q_0 až Q_{q-1}, symbolická vstupní písmena X_0 až X_{x-1}, symbolická výstupní písmena Y_0 až Y_{y-1}. Jazyk, umožňující popsat automat pomocí příkazu `switch`, je definován bezkontextovou gramatikou. Pro zachování možnosti definovat množinu vstupních a výstupních písmen a počet vnitřních stavů je gramatika parametrizována maximálním počtem vnitřních stavů *q*, počtem vstupních symbolů *x*, počtem výstupních symbolů *y*. Gramatika je dána následující definicí:

Množina terminálních symbolů:

T = { **`switch`**, `(`, `)`, **`state`**, `{`, `}`, **`case`**, `:`, **`if`**, `input`, `==`, `=`, `output`, **`break`**, `;`,`Q0`,…,`Qq-1`,`X0`,…,`Xx-1`,`Y0`,…,`Yy-1` }

Množina neterminálních symbolů:

N = { *START, S0, ..., Sn-1, P0, ..., Pm-1, NS, O* },

START je startovací symbol gramatiky, označení neterminálního symbolu *NS* je zkratkou anglického *next state*, jméno neterminálního symbolu *O* je zkratkou slova *output*.

Množina pravidel:

P = {

START → **`switch`**`(state){`*S0 S1 Sq-1* `}`

Si → **`case`** `Qi`: *P0 P1 Px-1*

Pi → **`if`** `(input==Xi) {state=`*NS*`; output=`*O*`; break; }`

NS → `Q0 | Q1 | ... | Qq-1`

O → `Y0 | Y1 | ... | Yy-1`

}

Gramatika generuje popis úplně určeného automatu. Chceme-li popsat neúplně určený automat, stačí přidat pravidlo typu *Pi* → *e*, kde *e* je prázdný řetězec.

Pro reprezentaci úplně určeného automatu s daným počtem stavů, vstupních písmen a výstupních písmen lze přesně vypočítat odpovídající délku kodonu. Délka *L* je rovna:

$$L = 3 \cdot q \cdot x + q \tag{8}$$

kde *q* je počet vnitřních stavů, *x* je počet vstupních písmen. Odvození vztahu není složité. Předpokládejme, že přepisujeme vždy levý neterminální symbol. Na pravé straně pravidla, přepisující startovací symbol *START*, je první neterminální symbol *S0*, pro jeho přepis se použije jeden gen. Následný gen se použije po náhradě *S0* pro přepis *P0* na podmínku `if`, dva geny pro přepis *NS* a *O*. Tedy, pro kompletní přepis každého pravidla *Pi* jsou potřeba 3 geny. V každém pravidle *Si* je *x* neterminálních symbolů a počet pravidel *Si* je *q*. Celkem $3 \cdot q \cdot x$ genů. K této hodnotě je třeba připočítat *q* genů pro přepis každého *Si*, *tedy* $3 \cdot q \cdot x + q$.

Pro neúplně určený automat se změní situace pouze pro přepis pravidel *Pi*. Je-li přepsáno pravidlo *Pi* na prázdný symbol, stačí pro přepis jeden gen. Pokud je vybráno pravidlo *Pi* → **`if`** `(input==Xi) {state=`*NS*`; output=`*O*`;` **`break;`** `}`, jsou pro celkový přepis pravidla nutné opět 3 geny. Předpokládáme-li rovnoměrné rozložení hodnot genů a dvě pravidla pro přepis *Pi*, každé z nich je použito s pravděpodobností 0,5, tedy s pravděpodobností 0,5 jsou použity na kompletní přepis neterminálního symbolu *Pi* 3 geny, s pravděpodobností 0,5 jeden gen. Opět, celkem přepisujeme $q \cdot x$ pravidel se symbolem *Pi* na levé straně. Střední počet genů pro přepis všech neterminálních symbolů *Pi* je roven $0{,}5 \cdot 3 \cdot q \cdot x + 0{,}5 \cdot q \cdot x = 2 \cdot q \cdot x$. Počet genů pro přepis *Si* je *q*.

Výsledně, pro neúplně určený automat je střední délka kodonu:

$$L = 2 \cdot q \cdot x + q \tag{9}$$

Při dekódování kodonu (přepis genotypu na fenotyp) je přepisován v derivaci vždy první neterminální symbol. Číslo pravidla, které bude vybráno pro přepis neterminálního symbolu N_i, se vypočítá jako (*hodnota_genu* mod *r*), kde *r* je počet pravidel, pomocí nichž lze neterminální symbol N_i přepsat. Je-li k dispozici pouze jedno pravidlo, výpočet se neprovádí, zpracovávaný gen je přeskočen a toto pravidlo je automaticky aplikováno.

Pokud je počet pravidel k přepisu menší než délka kodonu (v případě neúplně určeného automatu), zbytek kodonu je ignorován. Je-li kodon kratší, po vyčerpání všech genů se při dekódování pokračuje od prvního genu.

Příklad 1:

Uvažujme plně definovaný automat se šesti vnitřními stavy Q_0 – Q_5, dvěma vstupními symboly X_0, X_1 a dvěma výstupními symboly Y_0, Y_1. Délka kodonu je 42 genů. Pravidla typu $P_i \rightarrow e$ nejsou uvažována. Gramatika je specifikována:

T = { `switch, (, ), state, {, }, case, :, if, input, ==, =, output, break, ;,Q0,…,Q5,X0,X1,Y0,Y1` }

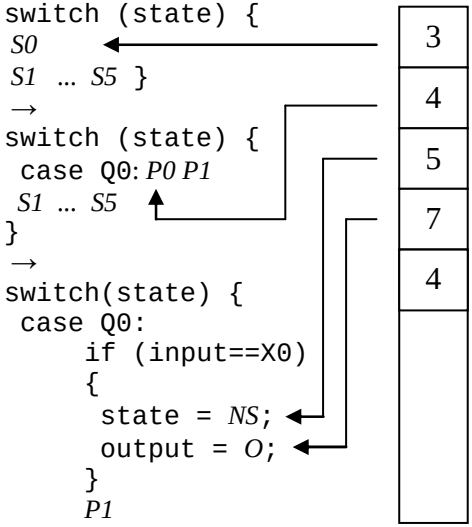
N = {*START, S0, S1, ..., S5, P0, P1, NS,O*}

Množina pravidel:
P = { *START* → **`switch`**`(state){` *S0 S1 … S5* `}`
S0 → **`case`** `Q0:` *P0 P1*
S1 → **`case`** `Q1:` *P0 P1*
atd.
P0 → **`if`** `(input==X0) {state=`*NS*`; output=`*O*`;`
`break``;}`
P1 → **`if`** `(input==X1) {state=`*NS*`; output=`*O*`;`
`break``;}`
NS → `Q0|Q1|Q2|Q3|Q4|Q5`
O → `Y0|Y1`
}

Ukázka dekódování kodonu na jazykový popis je na obrázku 3.

Protože je automat plně definován, neterminální symbol *S0* je automaticky přepsán podle pravidla `case Q0:` *P0 P1* ... a neterminální symbol *P0* je přepsán aplikací pravidla `if (input == X0) { state=` *NS*`; output =` *O*`; break;}`. Hodnota třetího genu je 5 a je použita pro výběr pravidla pro přepis neterminálu *NS* dle výpočtu: počet pravidel pro přepis

neterminálního symbolu *NS* je 6, vyhodnotíme (5 mod 6) = 5 a páté pravidlo *NS* → Q5 je aplikováno (pravidla jsou číslována od nuly). *NS* je přepsáno na Q5. Počet výstupních symbolů je 2, tedy i počet pravidel pro přepis neterminálního symbolu *O* je 2, analogicky (7 mod 2) = 1, pravidlo s pořadovým číslem 1 *O* → Y1 je vybráno a *O* je přepsáno na `Y1`.



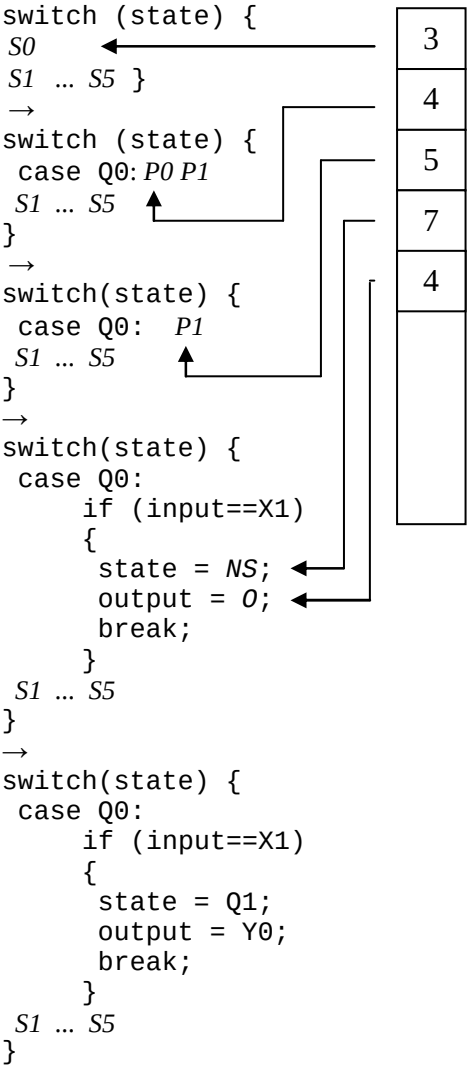
Obrázek 3: Dekódování kodonu z příkladu 1

Příklad 2:

Uvažujme automat se stejnými vlastnostmi jako v příkladu 1, ale neúplně určený. Do množiny pravidel přidáme dvě pravidla: $P_0 \rightarrow e$ a $P_1 \rightarrow e$, tedy:

P0 → *e* | `if (input==X0) {state=`*NS*`; output=`*O*`;`
`break; }`
P1 → *e* | `if (input==X1) {state=`*NS*`; output=`*O*`;`
`break; }`

Demonstrujeme dekódování stejného kodonu (obrázek 4). Stejně jako v prvém příkladu, pro přepis neterminálního symbolu *S0* existuje jedno pravidlo, *S0* je tedy automaticky přepsán podle pravidla `case Q0:` *P0 P1* ... Pro přepis neterminálu *P0* se využije hodnota genu 4: počet pravidel pro přepis je tentokrát roven 2, (4 mod 2) = 0, použijeme pravidlo číslo 0, *P0* → *e*; přechod pro vstupní písmeno X0 ze stavu Q0 je nedefinován. Dále postupujeme analogicky. Následující gen má hodnotu 5, na výběr pro přepis neterminálního symbolu *P1* jsou dvě pravidla, (5 mod 2) = 1. Vybíráme pravidlo číslo 1. Hodnota čtvrtého genu je použitého pro výběr pravidla pro přepis neterminálu *NS* je 7. Počet pravidel je tentokrát roven 6, vypočítáme (7 mod 6) = 1 a první pravidlo *NS* → Q1 je použito. Neterminální symbol *O* přepíšeme na Y0, protože (4 mod 2) = 0 a nulté pravidlo má podobu *O* → `Y0`.

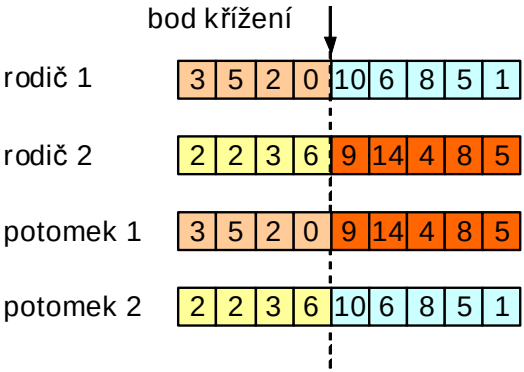


Obrázek 4: Dekódování kodonu z příkladu 2

3.1.1 Genetické operátory

Křížení

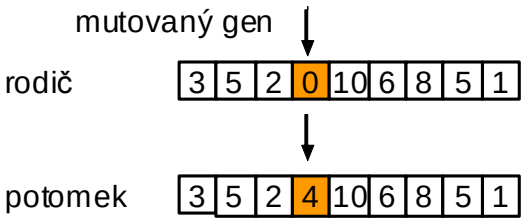
V algoritmu je použito jednobodové křížení s náhodně generovaným bodem křížení s rovnoměrným rozložením. Aplikací operátoru křížení nemůže vzniknout neplatný jedinec.



Obrázek 5: Operace křížení

Mutace

Operátor mutace vybírá náhodně gen a náhodně mění jeho hodnotu; změna hodnoty je provedena generováním nové celočíselné hodnoty.



Obrázek 6: Operace mutace

3.2 Experiment

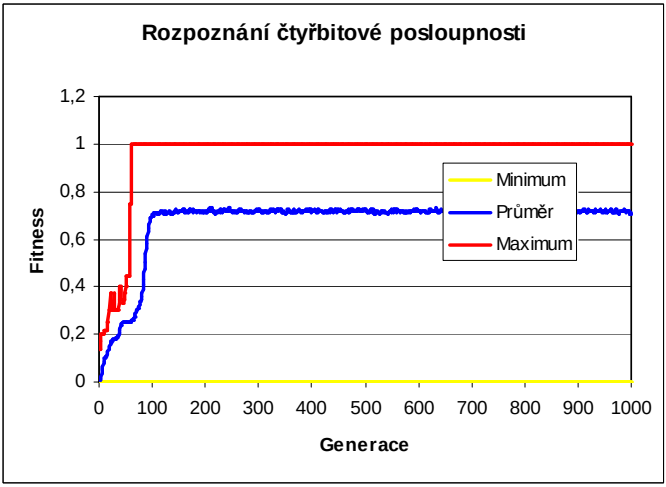
Výsledky testování gramatické evoluce na úloze U9 společně s porovnáním se standardním genetickým algoritmem jsou zachyceny v následujících grafech a tabulkách. Množina parametrů standardního genetického algoritmu odpovídala čtvrté kombinaci parametrů pravděpodobností křížení a mutace, parametry gramatické evoluce byly pouze v jedné kombinaci, viz tabulka 2. Společné parametry byly konstantní pro všechna spuštění algoritmu: velikost generace N = 10 000, maximální počet iterací I = 1 000, poměr *pom* byl počítán metodou PARTIAL. Oba automaty mají shodnou množinu vstupních a výstupních symbolů, maximální počet vnitřních stavů byl omezen na osm jako u předchozích úloh. Parametrizace M2 označuje parametry výpočtu skóre úspěšnosti přechodů u pravděpodobnostního algoritmu mutace: a = 1,0, d = 0,2, viz kapitola 2.4.

Algoritmus	**Pravděp. křížení *Pc***	**Pravděp. mutace Pm**	**Mutace s pravděp. algoritmem**
Genetický algoritmus	0,2	0,8	ano (parametrizace M2)
Gramatická evoluce	0,8	0,2	---

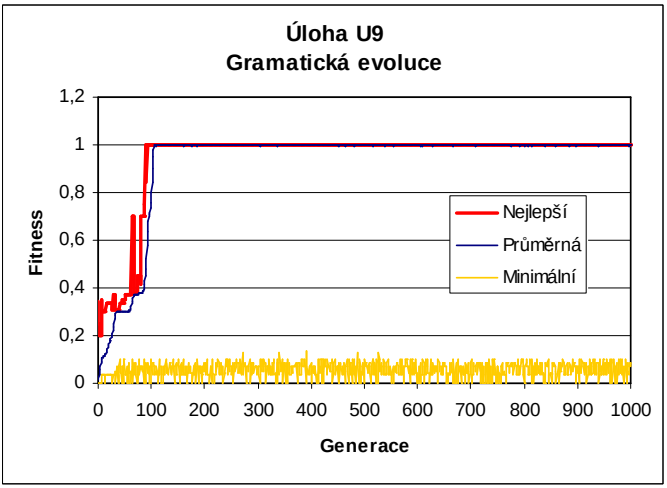
Tabulka 2: Parametry genetického algoritmu a gramatické evoluce

Správné řešení nalezly jak genetický algoritmus, tak gramatická evoluce. Průběh vývoje nejlepších hodnot a průměrných hodnot fitness funkce v generaci jsou na obrázku 7 a obrázku 8. Podstatné je, že čas běhu gramatické evoluce je kratší (tabulka 3).

Algoritmus byl testován za stejných podmínek jako předchozí experimenty: na notebooku HP Compaq nx9030 s procesorem Intel Pentium 1,6 GHz, 512 MB RAM. Při experimentu se snížením velikosti generace na 1000 jedinců nebylo správné řešení nalezeno.



Obrázek 7: Vývoj hodnot fitness funkce genetického algoritmu úlohy U9



Obrázek 8: Vývoj hodnot fitness funkce gramatické evoluce úlohy U9

Algoritmus	Čas běhu	Nalezeno řešení
Genetický algoritmus	**25 min. 49 sec.**	**ano**
Gramatická evoluce	**1 min. 25 sec.**	**ano**

Tabulka 3: Časy běhu algoritmů

3.3 Implementace

Algoritmus gramatické evoluce byl implementován v C++ jako konzolová aplikace. Aplikace se spouští s devíti povinnými parametry, jeden je nepovinný:

```
ge_fsm_2 vst_s X Y Q Pc Pm N I met [-l log_soubor]
```

kde,
vst_s ... jméno vstupního souboru s testovacími vektory
X ... počet vstupních písmen
Y ... počet výstupních písmen
Q ... max. počet vnitřních stavů
Pc ... pravděpodobnost křížení 0.0 - 1.0
Pm ... pravděpodobnost mutace 0.0 - 1.0
N ... velikost jedné generace
I ... počet iterací (generací)
met ... metoda v kriteriální funkci (ALL, PART)
log_soubor ... jméno výstupního souboru pro uložení statistiky (nepovinný)

Reprezentace kodonu a automatu je provedena třídami `TKodon` a `TmealyhoAutomaton`.

4. Závěr

Genetické algoritmy a jiné evoluční techniky jsou velice vyhledávanou technikou pro hledání optimálních řešení obtížných problémů. Snaha automatizovat návrh software a hardware i na té nejvyšší úrovni vedla ke genetickému programování, které evolučně tvoří programy v nějakém jazyce. Protože základním modelem chování v oblasti hardware a dnes i v oblasti software je konečný automat, vyplatí se věnovat pozornost aplikacím evolučních algoritmů na problém konstrukce konečného automatu.

Experimenty s konstrukcí konečného algoritmu pomocí genetického algoritmu ukázaly některé vlastnosti, např., že algoritmus vykazuje lepší chování, zvýší-li se pravděpodobnost operace mutace, což posouvá algoritmus k technikám typu evoluční strategie. Pozitivní je, pokud kriteriální funkce hodnotí nikoliv poměr splnění celých posloupností, ale postupně jejich počátečních podposloupností. Výrazné zlepšení vykazuje použití pravděpodobnostního operátoru mutace, bohužel, za cenu vzrůstu času výpočtu.

Proto je práce věnována zcela novému přístupu ke konstrukci automatu, a to pomocí gramatické evoluce. Gramatická evoluce je alternativou ke genetickému programování, ale má blíže ke genetickým algoritmům. Generuje program na základě gramatiky. Chování konečného automatu je možné popsat jazykově, pomocí cyklu, v jehož těle je příkaz switch, který softwarově konečný automat implementuje. Syntaxe příkazu switch je popsána bezkontextovou gramatikou, je možné tedy nechat pomocí gramatické evoluce takový jazykový popis konstruovat. Tato technika reprezentuje primárně automat lineárním jednorozměrným vektorem (kodonem), na rozdíl od klasické maticové dvourozměrné reprezentace. Experimenty prokázaly dobrou konvergenci algoritmu na úlohu rozpoznání čísel v bitové posloupnosti, kde podstromem grafu přechodů je strom, a oproti původnímu genetickému algoritmu s pravděpodobnostním operátorem mutace i snížení času běhu programu. Tato oblast si zasluhuje určitě další rozvoj - i vzhledem k tomu, že dnešní návrh číslicových obvodů nespočívá již v kreslení schématu sekvenčního obvodu, ale právě v popisu chování konečného automatu nejčastěji v jazycích VHDL nebo Verilog, tedy v jazykovém popisu tak, jak je generován gramatickou evolucí.

Literatura

[1] Fogel L.J., Owens A.J., Walsh M.J.: Artificial Intelligence Through Simulated Evolution, Wiley, New York, 1966

[2] Holland J. H.: Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975

[3] Lucas S. M., Reynolds T. J.: Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 27, No. 7, July 2005

[4] Goldberg D. E.: Genetic Algorithm in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989

[5] Mařík a kol.: Umělá inteligence 3, Academia Praha, 2002

[6] Mařík a kol.: Umělá inteligence 4, Academia Praha, 2003

[7] Koza J. R.: Genetic Programming, MIT Press, Cambridge, MA, 1994

[8] Jáneš V., Douša J.: Logické systémy, skriptum FEL ČVUT, 1993

[9] Jáneš V., Bokr J.: Logické systémy, monografie, ČVUT, 1999

[10] Langdon W.B, Poli R.: Foundations of Genetics Programming, Springer-Verlag 2002

[11] Ajzerman, M. A. a kol : Logika, automaty a algoritmy, Akademia Praha, 1971

[12] Melichar B.: Jazyky a překlady, skriptum FEL ČVUT, 2003

[13] Fábera V.: Využití genetických algoritmů ve flexibilních HW a SW strukturách - Konstrukce automatu genetickým algoritmem, doktorská disertační práce, ČVUT, 2007

[14] Fábera V., Jáneš V., Jánešová M.: Automata Construct with Genetic Algorithm. In 9th Euromicro Conference on Digital

System Design. Los Alamitos: IEEE Computer Society, 2006, p. 460-463. ISBN 0-7695-2609-8

[15] Chytil M.: Automaty a gramatiky, SNTL Praha, 1984

[16] Oplatková Z., Ošmera P., Šeda M., Včelař F., Zelinka I.: Evoluční výpočetní techniky - principy a aplikace, BEN, 2008, ISBN: 80-7300-218-3

[17] Korček P., Sekanina L., Fučík O.: Evolutionary approach to calibration of cellular automaton based traffic simulation model, In: Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems, Anchorage, US, IEEE ITSS, 2012, pp. 122-129, ISBN 978-1-4673-3062-6

[18] O'Neil M., Ryan C.: Grammatical Evolution, IEEE Transaction on Evolutionary Computation, Vol. 5, Num. 4, 2001

[19] Dempsey I., O'Neill M., Brabazon A: Foundations in Grammatical Evolution for Dynamic Environments, Springer, 2009

Profesní životopis - Ing. Vít Fábera, Ph.D.

Dosažené vzdělání:

1989 – 1993 Gymnázium Milevsko,
Masarykova 183, 399 01 Milevsko

1993 – 1999 Ing., obor Informatika a výpočetní technika
České vysoké učení technické v Praze,
Fakulta elektrotechnická
Technická 2, 160 00 Praha 6

2001 – 2007 Ph.D., obor Inženýrská informatika v dopravě a spojích
České vysoké učení technické v Praze,
Fakulta dopravní
Konviktská 20, 110 00 Praha 1

Pracovní pozice:

Od roku 2001 působím jako odborný asistent na Ústavu aplikované informatiky v dopravě (dříve Ústav informatiky a telekomunikací), dále jako zkušební technik a manažer kvality akreditované Zkušební laboratoře Fakulty dopravní a manažer kvality Certifikačního orgánu pro výrobky při Fakultě dopravní ČVUT.

Výuka:

Přednáším a cvičím předměty z oblasti programování (Základy informatiky, Úvod do programování, Programovací jazyk C/C++, Programování I/II), jednočipových mikropočítačů (Jednočipové mikropočítače), podílím se na výuce předmětu Hardware počítačů a Základy číslicové techniky (přednášky i cvičení) a dále na výuce předmětu Řízení v komplexních systémech. V řádném studiu jsem zavedl předměty:

Aplikace numerických metod (2+2)
Programování I
Programování II

Projekt „Evoluční techniky v dopravě“

Odborné zájmy:

Mezi moje odborné zájmy patří aplikace genetických algoritmů, teorie automatů, logické obvody a jejich implementace pomocí hradlových polí FPGA, mikroprocesorová technika, obecně programování (jazyk C/C++).

Ostatní aktivity:

Byl jsem členem řešitelského týmu grantu „Dynamika systémových aliancí“ IAA 201240701, jehož výsledky AV ČR ocenila jako vynikající, nyní jsem spoluřešitelem grantu „Vývoj motorové lokomotivy poháněné CNG“ MPO FR-TI3/575 CNG \ 1ND. V rámci tohoto grantu se zabývám tvorbou software pro přenos dat pod OS Linux a tvorbou řídicích algoritmů pro jednočipové mikropočítače.
Ve zkušební laboratoři kromě provádění zkoušek tvořím software pro interní mezikalibrační kontroly, jako manažer kvality jsem tvůrcem interních postupů (jak ve zkušební laboratoři, tak v certifikačním orgánu).

Publikace:

Samostatné části v mezinárodní monografii

Votruba Z., Novák M., Brandejský T., Fábera V., Bouchner P., Zelenka J., Vysoký P., Bělinová Z., Sadil J.: Theory of System Alliances in Transportation Science, monografie, Neural Network Word, 2009, ISBN 978-80-87136-08-9, spoluautorem kapitol Multilingual approach to system alliances (podíl 80%), Reliability aspects (podíl 20%)

Článek v impaktovaném časopise

Fábera V., Jáneš V., Jánešová M., Zelenka J.: Regular Grammar Transformation Inspired by the Graph Distance Using GA, Neural

Network Word, 2011, vol. 21, ISSN 1210-0552, podíl 80%

Článek v mezinárodním recenzovaném časopise

Fábera V.: Theory of FSMs Sharing Internal States, In International Journal of Computers and Communications, Volume 7, 2013, NAUN, p. 13-17, ISSN: 2074-1294, podíl 100%

Příspěvek na mezinár. konferenci (ve sborníku)

Fábera, V.: Transformations of Pairs of FSMs. In LATEST TRENDS in INFORMATION TECHNOLOGY. Athens: WSEAS, 2012, p. 403-408. ISBN 978-1-61804-134-0, podíl 100%

Fábera V., Zelenka J., Jáneš V., Jánešová M.:, Grammatical Evolution and FSM Construction, In MENDEL 2012, pp. 94-99, Brno: VUT v Brně, Fakulta strojního inženýrství, 2012, ISBN 978-80-214-4540-6, podíl 70%

Přibyl P., Fábera V., Faltus V., Týfa L.: Domain Oriented Ontology for ITS Systems, Proceedings of 9th International Conference Elektro 2012, pp. 364 – 368, ISBN 978-1-4673-1178-6, podíl 25%

Fábera, V. - Zelenka, J.: FSM Construct with Genetic Algorithm Using Distance Measuring in Mutation Operator. In Proceedings of 17th International Conference on Soft Computing (MENDEL 2011). Brno: University of Technology, 2011, p. 62-66. ISBN 978-80-214-4302-0, podíl 50%

Fábera, V. - Jáneš, V. - Jánešová, M. - Pastor, O.: Implementation of MSC Decompression Algorithm in the Microblaze Processor. In Proceedings of the Work in Progress Session - DSD 2011. Oulu: University of Oulu, 2011, vol. 1, p. 3-4. ISBN 978-3-902457-30-1, podíl 25%

Fábera, V. - Jáneš, V. - Jánešová, M.: The Distance between FSMs and its Computing Using Genetic Algorithm. In Proceedings

of CSE 2010 International Scientific Conference on Computer Science and Engineering. Košice: Technická Univerzita, podíl 60%

Fábera, V. - Jáneš, V. - Jánešová, M.: Test of Genetic Algorithm with Fitness Measuring Distance between FSMs. In Proceedings of the Work in Progress Session SEAA 2010 and DSD 2010. Linz: Johannes Kepler University, 2010, p. 16-17. ISBN 978-3-902457-27-1, podíl: 60%

Fábera, V. - Zelenka, J.: Regular Grammar Transformation Using Graph Distance Computed by GA. In Mendel 2010. Brno: University of Technology, 2010, p. 86-90. ISBN 978-80-214-4120-0, podíl 50%

Fábera, V. - Jáneš, V. - Jánešová, M.: Transformation of Pair of FSMs Sharing Input Symbols. In Proceedings of the Work in Progress Session SEAA 2009 and DSD 2009. Linz: J. Kepler University - FAW, 2009, p. 25-26. ISBN 978-3-902457-25-7, podíl: 60%

VŠ učebnice

Fábera, V. - Krušina, K. - Malinovský, V.: *Sbírka řešených úloh z programování v jazyku C*. 1. vyd. Praha: ČVUT, 2009. 152 s., ISBN 978-80-01-04451-3, podíl: 33,3%

VŠ skripta

Malinovský, V. - Fábera, V. - Malý, K. - Krušina, K.: *Základy informatiky*. 1. vyd. Praha: ČVUT, Fakulta dopravní, 2011, 156 s., ISBN 978-80-01-04745-3, podíl: 25%

Původní příspěvek na české konferenci:

Brandejský, T. - Leso, M. - Fábera, V.: Aplikace multicore procesorů a multiprocesorových systémů - analýza rizik. In 17. medzinárodné sympózium "Zvyšovanie konkurencieschopnosti európského železničného systému" EURO - Žel 2009. Zborník prednášok. Žilina: Žilinská univerzita, 2009, díl 1, s. 179-184. ISBN 978-80-554-0024-2, podíl 20%

Fábera V., Stambolidis M.: Solving VRP Using Genetic Algorithm with Time Constraints in Fitness Function, Proceedings of Multidisciplinary Academic Conference of Transport, Logistics and Information Technologies, 2013, ISBN 978-80-905442-0-8, podíl 50%