VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Jednoduchý nástroj pro ověřování ekvivalence systémů: Rozhodování bisimilarity mezi nBPA a nBPP

Simple Equivalence Checking Tool: Deciding Bisimilarity of nBPA and nBPP

2014 Bc. Martin Bradáč

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Bradáč**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Jednoduchý nástroj pro ověřování ekvivalence systémů: Rozhodování bisimilarity mezi nBPA a nBPP
Simple Equivalence Checking Tool: Deciding Bisimilarity of nBPA and nBPP

Zásady pro vypracování:

Cílem práce je sestrojit jednoduchý nástroj pro ověřování bisimulační ekvivalence mezi dvěma systémy, z nichž jeden je specifikován jako normovaný základní paralelní proces a druhý jako normovaná základní procesní algebra. Příslušný algoritmus je složen z několika menších podprogramů majících význam i samostatně a všechny jsou popsány v jednom anglicky psaném článku slovně nebo pseudokódem. Jednotlivé podprogramy budou vysvětleny i vedoucím práce na konzultacích a student má za úkol je jen implementovat.

Postup práce:
1. Nastudovat pojmy Basic Parallel Processes (BPP, základní paralelní procesy), Basic Process Algebra (BPA, základní procesní algebry) a bisimulační ekvivalence a popis požadovaného algoritmu.
2. Vytvořit nástroj rozhodující ekvivalenci mezi zadaným normovaným BPP a normovaným BPA systémem.
3. Nástroj bude co nejvíce modulární, aby se do něj daly v budoucnu doplňovat algoritmy rozhodující jiné ekvivalence nebo pracující s jinými modely apod.
4. Vstupní systémy bude moci vhodným způsobem zadat uživatel, ale taky bude k dispozici generátor náhodných vstupů podle parametrů (počet proměnných, abeceda atd.) zvolených uživatelem.
5. Součástí práce bude i porovnání teoretické časové složitosti daného algoritmu (ta je v článku uvedena) a prakticky zjištěných časů běhu na vygenerovaných nebo zadaných vstupech.

Seznam doporučené odborné literatury:

Petr Jančar, Martin Kot, Zdeněk Sawa: Normed BPA vs. Normed BPP Revisited
in: Proceedings of CONCUR 2008, Lecture Notes in Computer Science 5201, Springer, 2008, ISBN 978-3-540-85360-2, pages 434-446, http://www.springerlink.com/content/r5344n418p3l11v1/

Další podle pokynů vedoucího práce

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v diplomových programech VŠB-TU Ostrava.

V Ostravě dne 25. 4. 2014 ...

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 25. 4. 2014 ...

Rád bych poděkoval svému vedoucímu práce Ing Martinu Kotovi, Ph.D. za odborné vedení, užitečné rady, pomoc a věnovaný čas při tvorbě této diplomové práce.

Abstrakt

Ověřování ekvivalencí náleží mezi ustálené techniky verifikace systému. Ověřují se různé ekvivalence na odlišných modelech systému. Cílem diplomové práce bylo vytvoření jednoduchého verifikačního nástroje, který ověřuje bisimulační ekvivalenci mezi normovanými základními paralelními procesy a normovanými základními procesními algebrami, tedy mezi dvěma různými modely.

V tomto dokumentu je objasněno vytvoření tohoto nástroje, počítaje v to i vysvětlení pojmů, včetně jejich vlastností, nezbytných k porozumění popsaných algoritmů. Rozbor a věcná správnost algoritmů určení jejich časové složitosti plyne z citovaných zdrojů, tento text s tímto nepracuje.

Klíčová slova

jednoduchá procesní algebra, základní paralelní procesy, bisimulační ekvivalence, bisimilarita, BPA, BPP, verifikace, verifikační nástroj

Abstract

Checking the equivalences belongs among settled techniques of verifications of the systems. Different equivalences are checked on different models of the system. The aim of my thesis was the creation of the simple verification tool which would be checking bisimulation equivalences between normed basic parallel processes and normed basic process algebras, so between two of these models.

In this document is described how the tool was created, including explanation of notion and their properties. This is necessary for understanding described algorithms. Determination of time complexity comes from quoted sources, this text does not deal with it.

Key words

basic process algebra, basic parallel processes, bisimulation equivalence, bisimilarity, BPA, BPP, verification, verification tool

Seznam použitých symbolů a zkratek

BPP	–	základní paralelní procesy
	–	basic parallel process
BPA	–	jednoduchá procesní algebra
	–	basic process algebra
nBPP	–	normovaný BPP
nBPA	–	normovaný BPA
LTS	–	ohodnocený přechodový systém
	–	labelled transition system
SSK	–	silně souvislé komponenty
SF	–	single final

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

Seznam grafů

1. Úvod

Verifikace je proces ověřování, zda implementace odpovídá specifikaci. Nejznámější techniky verifikace jsou testování a simulace. Tyto techniky jsou efektivně používány během téměř všech fází vývoje. Jejich velkou nevýhodou je, že ve většině případů nemohou zaručit bezchybnost v každé možné situaci. Počet vstupů je obvykle velký, často nekonečný, tudíž není možné vyzkoušet všechny možnosti během testování a simulací.

Další technika je formální verifikace, která prochází všechna možná chování pro zajištění bezchybnosti, dělí se na tři skupiny: ověřování ekvivalencí (equivalence checking), ověřování modelů (model checking), dokazování vět (theorem proving). V diplomové práci využíváme techniku ověřování ekvivalence, které zkoumá, zda dva systémy jsou v nějakém smyslu ekvivalentní.

Tématem této diplomové práce je vytvoření jednoduchého nástroje, jehož úkolem je ověřování bisimulační ekvivalence mezi dvěma systémy a to konkrétně mezi normovaným základním paralelním procesem a normovanou základní procesní algebrou.

V kapitole 2 jsou objasněny základní pojmy systémů, včetně jejich vlastností. Dále zde nalezneme vysvětlení algoritmů použitých ke zjištění určitých vlastností.

Kapitola 3 se zaměřuje na modelování a analýzu systému. Také popisuje požadované chování systému, jak nástroj využívat z pohledu uživatele.

Klíčovým bodem je kapitola 4, ve které je nejprve vysvětlena struktura a funkčnost jednotlivých komponent systému, následně je zaměřena na ověřování bisimulační ekvivalence a to za podpory dvou základních algoritmů.

Kapitola 5 je rezervována na práci se vstupními systémy BPP a BPA a to ve třech možnostech – vlastní zadání, načtení ze souboru a vygenerování náhodného vstupu.

Následující předposlední kapitola 6 popisuje časovou náročnost výpočtů při ověřování bisimulační ekvivalence se vstupy různé velikosti, realizované na třech rozdílných zařízeních. Výstupem kapitoly pak budou získané výsledky včetně jejich vyhodnocení.

Závěrečná kapitola 7 rekapituluje celkové výsledky a přínosy práce.

1.1. Cíle práce

Prvním cílem je zhotovit účinný nástroj, který bude ověřovat bisimulační ekvivalenci mezi dvěma systémy. Jedním zadaným ve formě normovaného základního paralelního procesu a druhým ve formě normované základní procesní algebry.

Druhým cílem je, aby uživatel mohl vhodným způsobem zadávat vstupní systémy jak manuálně, tak i s pomocí generátoru náhodných vstupů zvolených parametrů.

Závěrečným cílem je pak komparace vstupů odlišných velikostí při ověřování (výpočtů) bisimulační ekvivalence a to z hlediska časové složitost.

2. Teorie

V této kapitole jsou vysvětleny základní pojmy, definice používaných systémů, jejich vlastnosti. Dále také vysvětlení algoritmů použitých ke zjištění určitých vlastností.

2.1. Ohodnocené přechodové systémy – LTS

Základním konceptem pro popis systému, který ověřuje bisimulační ekvivalenci, jsou takzvané ohodnocené přechodové systémy (Labelled Transition Systems – LTS, [1]). LTS jsou formálně definovány jako trojice $(S, A, \rightarrow)$, kde:

- S je (potencionálně nekonečná) množina stavů,
- A je konečná množina akci,
- $\rightarrow \subseteq S \times A \times S$ je přechodová relace.

Neboli můžeme říct, že S je (potencionálně nekonečná) množina všech možných stavů systému. A je množina navenek pozorovatelných akci, které má daný systém. Přechodová relace představuje chování systému.

Místo $(s, a, s') \in \longrightarrow$ používáme $s \xrightarrow{a} s'$, což můžeme číst jako „systém, ve stavu s, může provést akci a a přejít do stavu s“ nebo taky „systém ve stavu s reaguje na akci a přechodem do stavu s“. Stav s označujeme, jako vstupní stav a stav s' jako výstupní stav. Zápis $s \xrightarrow{a} s'$ může být rozšířen na posloupnost akcí $w = a_1, a_2, \ldots, a_n$, kde $w \in A^*$. Poté píšeme $s \xrightarrow{w} s'$ právě tehdy, když existuje posloupnost stavů $s_0, s_1, \ldots, s_n$ taková, že $s_0 = s, s_n = s'$ a $s_{i-1} \xrightarrow{a_i} s_i$ pro každé i takové, že $1 \leq i \leq n$. Stav s' je dosažitelný ze stavu s (psáno $s \longrightarrow {}^* s'$) právě tehdy, když existuje nějaké $w \in A^*$ takové, že $s \xrightarrow{w} s'$.

Množina stavů PRE označuje všechny vstupní stavy množiny přechodů nebo jednoho přechodu. Množina stavů POST označuje všechny výstupní stavy množiny přechodů nebo jednoho přechodu. Máme-li přechod $t = s \xrightarrow{a} s'$, pak $PRE(t) = \{s\}$ a $POST(t) = \{s'\}$.

Příklad:

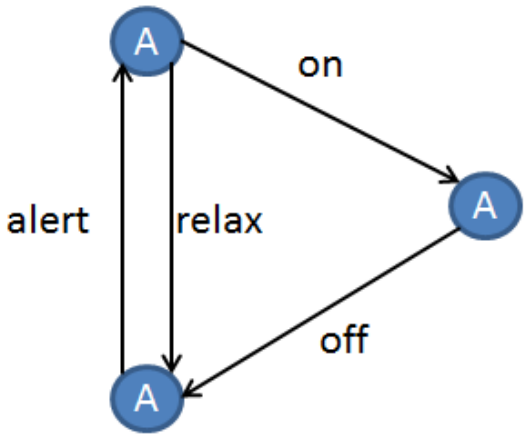
Mějme následující LTS:

$S = \{I, R, A\}$

$A = \{alert, relax, on, off\}$

$\rightarrow = \{(I, alert, R), (R, relax, I), (R, on, A), (A, off, I)\}$

Jednoduchý LTS se třemi stavy, čtyřmi akcemi a čtyřmi přechodovými relacemi je zobrazen na obrázku 1.



Obrázek 1: Grafické zobrazení jednoduchého LTS

2.2. Petriho sítě

Pro vysvětlení dalších vlastností systémů lze využít Petriho sítě (Petri nets, PN-systém, [2], [3]), definované jako pětici (P, T, I, O, M_0), kde:

- P je konečná množina míst
- T je konečná množina přechodů, $T \cap P = \emptyset$
- I je zobrazení typu $T \rightarrow P_{MS}$, vstupní funkce
  - P_{MS} je množina všech multimnožin nad množinou P.
- O je zobrazení typu $T \rightarrow P_{MS}$, výstupní funkce
  - P_{MS} je množina všech multimnožin nad množinou P.
- M_0 je zobrazení typu $P \rightarrow N$, tzv. počáteční značení, kde $N = \{0,1,2, \dots\}$ je množina přirozených čísel.

2.2.1. Dosažitelnost

Množina dosažitelnosti (reachablility set) Petriho sítě (P, T, I, O, M_0) je množina $RS(M_0)$, definovaná takto:

- $M_0 \in RS(M_0)$
- $M \in RS(M_0) \wedge (\exists t \in T)\{Mt > M'\} \Longrightarrow M' \in RS(M_0)$

Značení M' PN-systému je dosažitelné (reachable) ze značení M, jestliže existuje posloupnost přechodů σ_M, která je proveditelná ve značení M a která převádí LTS ze značení M do značení M'.

2.2.2. Omezenost

Místo *p* PN-systému se nazývá *k*-omezené (*k*-bounded), jestliže pro každé dosažitelné značení je počet tokenů v tomto místě nanejvýše rovný *k*, tj.

$$\big(\forall M \in RS(M_0)\big)[M(p) \leq k]$$

PN-systém se nazývá k-omezený, jestliže všechna jeho místa jsou *k*-omezená, tj. platí-li

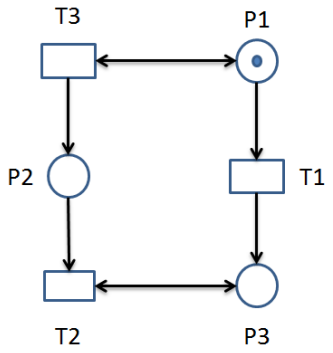
$$(\forall p \in P)(\forall M \in RS(M_0)[M(p) \leq k]$$

2.2.3. Graf pokrytí

Je dáno značení PN-systému $M = (m_1, m_2, \dots, m_{|P|})$, kde $|P|$ je počet prvků množiny M. Může-li počet tokenů v místě i růst do nekonečna, potom zapisujeme $m_i = \omega$. Graf pokrytí vychází ze stromu pokrytí (viz [3]), který se konstruuje následovně:

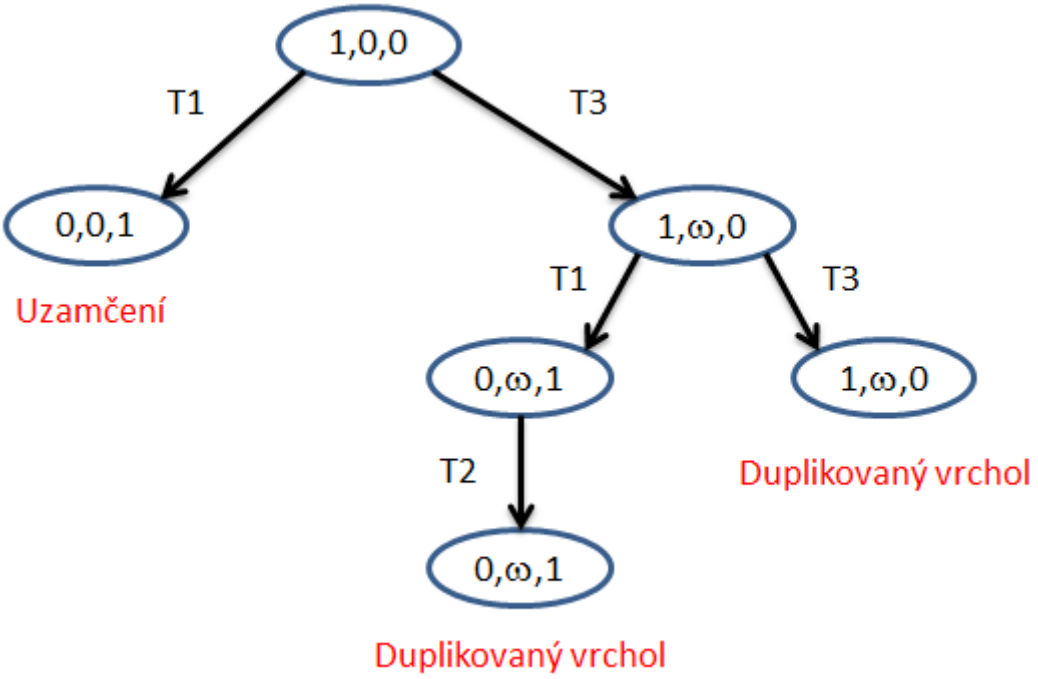
1) Kořenem stromu pokrytí je počáteční značení M_0.
2) Pro každé nově ustanovené značení M platí:
   - Je-li $E(M) = \emptyset$, pak je M koncovým vrcholem stromu.
   - Není-li $E(M) = \emptyset$, pak nalezni všechna bezprostředně dosažitelná značení M' reprezentující vrcholy grafu podle těchto pravidel:
     - Je-li $m_i = \omega$ pro některé, pak taky $m'_i = \omega$
     - Jestliže v dosud vytvořeném stromě na cestě od kořene M_0 do vrcholu M' existuje vrchol N, jehož značení je pokryto značením vrcholu M', potom polož $m'_i = \omega$ pro všechna $i \in \{1,2, \dots, |P|\}$, taková, že $m'_i > n_i$
     - V ostatních případech se značení vrcholu M' vypočte podle pravidla pro provedení přechodu
3) Rozšiřování stromu je ukončeno, jakmile jsou všechny koncové vrcholy buď uzamčením (tedy marking, ve kterém není uschopněna žádná akce) anebo se na své větvi stromu vyskytují již po druhé (duplikované vrcholy).

Příklad: Na obrázku 2 je dán PN-systém



Obrázek 2: PN-systém

Je zřejmé, že prováděním přechodu *T3* roste značení místa *P2* neomezeně. Provedením přechodu *T1* přestane být přechod *T3* proveditelný a opakovaným prováděním přechodu *T2* klesá značení místa *P2* až do značení (0,0,0), jež je uzamčení. Strom dosažitelnosti je v tomto případě nekonečný, proto ho nahradíme stromem pokrytí, viz obrázek 3.



Obrázek 3: Strom pokrytí

Graf dosažitelnosti vznikne ze stromu dosažitelnosti odstraněním koncových zdvojených uzlů a jejich náhradou zpětnými hranami jejich dvojníkům.

2.2.4. Silně souvislé komponenty – SSK

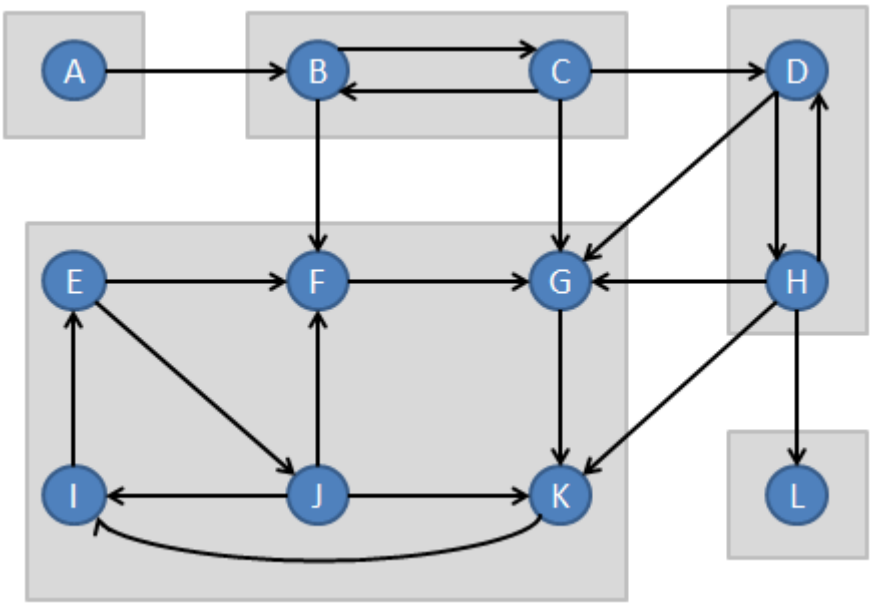
Můžeme říct, že neorientovaný graf G je souvislý, jestliže mezi každou dvojicí vrcholů existuje cesta. Jinými slovy, z každého vrcholu $u \in V(G)$ se dostaneme do libovolného jiného vrcholu $v \in V(G)$ po nějaké cestě. Graf G můžeme rozložit na komponenty souvislosti. To jsou největší podgrafy grafu G, které jsou souvislé (viz [5]).

V souvislém orientovaném grafu platí to stejné, ale po šipkách můžeme projít jen jedním směrem. Opět bychom chtěli zavést „souvislost" tak, aby bylo možné se po šipkách dostat z libovolného vrcholu $u \in V(G)$ do libovolného vrcholu $v \in V(G)$. Tentokrát se ale může stát, že orientovaná cesta z u do v povede jinudy než cesta z v do u.

Dva vrcholy $u, v \in V(G)$ jsou spolu silně propojeny, pokud existuje orientovaná cesta z u do v a zároveň orientovaná cesta z v do u. Orientovaný graf G nazveme silně souvislý, pokud je každá dvojice vrcholů $u, v \in V(G)$ silně propojena.

Relace „být silně propojen" je relace ekvivalence. Třídám této ekvivalence budeme říkat *silně souvislé komponenty.* Jinými slovy, silně souvislá komponenta je maximální podgraf grafu G, který je silně souvislý.

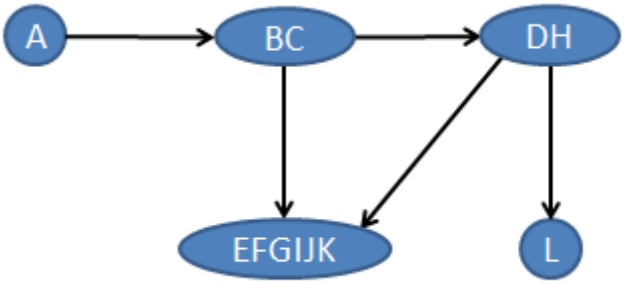
Příklad:



Obrázek 4: Silně souvislé komponenty

V grafu na obrázku 4 máme pět silně souvislých komponent vyznačených šedými čtyřúhelníky.

Kontrakcí každé silně souvislé komponenty do jednoho meta-vrcholu dostaneme meta-graf, viz obrázek 5.



Obrázek 5: Meta-graf silně souvislých komponent

Pozorování*: Meta-graf, který vznikl kontrakcí silně souvislých komponent je acyklický orientovaný graf.*

2.2.5. Nalezení SSK - Tarjanův algoritmus

Tarjanův algoritmus (Tarjan's algorithm, [6]) je grafový algoritmus. Slouží k vyhledávání silných komponent (viz kapitola 2.2.4) orientovaného grafu.

Tarjanův algoritmus je založen na prohledávání do hloubky. Vrcholy jsou indexovány podle pořadí svého nalezení. Při návratu z rekurze se každému vrcholu přiřadí uzel s nejnižším indexem, na jaký lze dosáhnout. Všechny vrcholy, které mají totožný cílový uzel (index), jsou ve stejné komponentě.

Tarjanův algoritmus má stejně jako prohledávání do hloubky asymptotickou složitost $O(|U| + |H|)$, kde *U* je množina uzlů a *H* je množina hran.

```
index = 0
/*
 * Spousti tarjanuv algoritmus
 * @param g graf, v nemz budou hledany silne komponenty
 * @return seznam komponent
 */
List executeTarjan(Graph g)
  Stack s = {}
  List scc = {} //seznam silnych komponent
  for Node node in g
      if (v.index is undefined)
          tarjanAlgorithm(node, scc, s)
  return scc

/*
 * Tarjanuv algoritmus
 * @param node zpracovavany uzel
 * @param SCC seznam komponent (seznamu uzlu v komponentach)
 * @param s zasobnik
 */
procedure tarjanAlgorithm(Node node, List scc, Stack s)
  v.index = index
  v.lowlink = index
  index++
  s.push(node) //pridej na zasobnik
  for each Node n in Adj(node) do //pro vsechny potomky
    if n.index == -1 //pokud jeste nebyl uzel objeven
      tarjanAlgorithm(n, scc, s, index) //prohledej
      node.lowlink=min(node.lowlink, n.lowlink) //uprav lowlink otce
    else if stack.contains(n) //pokud komponenta nebyla jiz uzavrena
      node.lowlink = min(node.lowlink, n.index)

  if node.lowlink == node.index //pokud jsme v koreni komponenty
    Node n = null
    List component //seznam uzlu dane komponenty
    do
      n = stack.pop() //vyber uzel ze zasobniku
      component.add(n) //pridej ho do komponenty
    while(n != v) //dokud nejsme v koreni
    scc.add(component) //komponentu pridej do seznamu komponent
```

Výpis kódu 1: Pseudokód Tarjanova algoritmu

2.3. Bisimulační ekvivalence

Nechť $(S, A, \longrightarrow)$ je LTS. Binární relace $R \subseteq S \times S$ je bisimulace právě tehdy, když pro každou dvojici stavů $(s, t) \in R$ a každou akci $a \in A$ jsou splněny následující podmínky:

- Jestliže existuje nějaké $s' \in S$ takové, že $s \xrightarrow{a} s'$, pak musí existovat nějaké $t' \in S$ takové, že $t \xrightarrow{a} t'$ a $(s', t') \in R$.
- Jestliže existuje nějaké $t' \in S$ takové, že $t \xrightarrow{a} t'$, pak musí existovat nějaké $s' \in S$ takové, že $s \xrightarrow{a} s'$ a $(s', t') \in R$.

Poznámka: Toto nám říká, že $s \xrightarrow{a} s'$ odpovídá $t \xrightarrow{a} t'$, respektive $t \xrightarrow{a} t'$ odpovídá $s \xrightarrow{a} s'$.

Stavy s, t jsou bisimulačně ekvivalentní (bisimilární), psáno $s \sim t$, právě tehdy, když existuje nějaká bisimulace R taková, že $(s, t) \in R$. Relace $\sim$ se nazývá bisimulační ekvivalence neboli bisimilarita. Viz [1].

Bisimulační ekvivalence na LTS $(S, A, \longrightarrow)$ může být alternativně popsána z hlediska podmínek bisimulační hry, kterou hrají dva hráči označení jako útočník a obránce. Pozice ve hře jsou dvojice $(s, t) \in S \times S$. Hra probíhá v kolech. V pozici (s, t) si útočník vybere jeden stav z dvojice s a t (řekněme s) a některý přechod vedoucí z tohoto vybraného stavu (řekněme $s \xrightarrow{a} s'$). Obránce poté vybere některý přechod z druhého stavu se stejným ohodnocením, které vybral útočník (řekněme $t \xrightarrow{a} t'$). Hra pokračuje v dalším kole v pozicích (s', t'). Pokud některý z hráčů uvázne v některé pozici ve hře, tedy nemá žádnou další možnost volby pohybu, vyhrává jeho soupeř. V případě, že je hra nekonečná, je považován za vítěze obránce.

Poznámka: $s \sim t$ právě tehdy, když v bisimulační hře startující v pozici (s, t) má obránce vítěznou strategii.

Přestože je bisimulační ekvivalence definována na stavech jednoho LTS, může být vždy použita také na stavech dvou různých LTS, díky možnosti jejich disjunktního sjednocení.

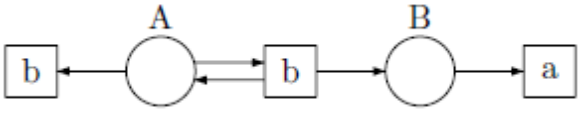
2.4. Základní paralelní procesy BPP

Základní paralelní procesy (Basic Parallel Processes – BPP, [7]) mohou být definovány následovně: BPP systém Δ je, vzhledem k množině atomických akcí $Act = \{a, b, \dots\}$ a množině procesních proměnných $Var = \{X, Y, \dots\}$, konečná množina pravidel ve tvaru $X \xrightarrow{a} Y_1 Y_2 \dots Y_n$. Var_Δ je poté množina proměnných a Act_Δ množina akcí vyskytujících se v pravidlech Δ. Z BPP systému Δ může vzniknout LTS, které má Var_Δ^* jako stavy, Act_Δ jako akce, a kde $M \xrightarrow{a} M'$ právě tehdy, když existuje nějaké pravidlo $X \xrightarrow{a} \alpha$ v Δ takové, že $X \in M$ a $M' = (M - \{X\}) \cup mset(\alpha)$. $mset(\alpha)$ značíme multimnožinu prvků z α. Je to tedy funkce přiřazující ke každé proměnné X počet výskytů v α .

Příklad: Mějme následující gramatiku reprezentující BPP systém:

$Var = \{A, B\}$
$Act = \{a, b\}$
$A \xrightarrow{b} \varepsilon$
$A \xrightarrow{b} AB$
$B \xrightarrow{a} \varepsilon$

Grafické zobrazení tohoto systému je na obrázku 6.



Obrázek 6: Příklad BPP

Pro reprezentaci BPP systémů budeme využívat speciální typ Petriho sítě, kde má přechod právě jedno vstupní místo. BPP síť je n-tice $\Sigma = (P, T, PRE, POST, \lambda)$, kde

- P je konečné množina míst
- T je konečná množina přechodů
- $PRE: T \to P$ je funkce, která přiřazuje vstupní místo $PRE(t)$ všem přechodům t
- $POST: T \to P^\oplus$ ($P^\oplus$ je množina multimnožin nad P) je funkce, která přiřazuje množinu výstupních míst $POST(t)$ každému přechodu t, kde počet výskytů místa p v $POST(t)$ reprezentuje počet tokenů vkládaných do místa p přechodem t
- $\lambda: T \to Act$ je funkce přiřazující přechodům akce z Act

Používáme funkci $F: T \times P \to N$ takovou, že $F(t, p)$ je počet výskytů p v $POST(t)$, tj. $POST(t)(p)$. Hodnota $F(t, p)$ reprezentuje počet tokenů vkládaných přechodem t do místa p. $PostPlaces(t)$ značíme množinu výstupných míst přechodu, tj. $PostPlaces(t) = \{p \in P | F(t, p) > 0\}$. Definujeme $T(p) = \{t \in T | PRE(t) = p\}$.

Značení (Marking) M je množina míst, tj. prvek z $P^\oplus$. Množinu všech značení z Σ označujeme $M(\Sigma)$. Pro označení prázdného značení používáme M_ε, tj. značení M takové, že $\forall p \in P: M(p) = 0$. Místo p je označeno v značení M, pokud $M(p) > 0$, v opačném případě v něm označeno není. Přechod t je uschopněn v M, jestliže $M(PRE(t(\geq 1)))$. Uschopněný přechod může byt provedený s výsledkem $M' = (M - \{PRE(t)\}) \cup POST(t)$, poté píšeme $M \xrightarrow{t} M'$.

BPP síť $\Sigma = (P, T, PRE, POST, \lambda)$ definuje LTS s množinou stavů $M(\Sigma)$, abecedou Act a přechodovou relaci, kde $M \xrightarrow{a} M'$ právě tehdy, když $M \xrightarrow{t} M'$ pro nějaké $t \in T$ takové, že $\lambda(t) = a$. BPP proces je dvojice (M_0, Σ), kde Σ je BPP systém a $M_0 \in M(\Sigma)$ je počáteční značení.

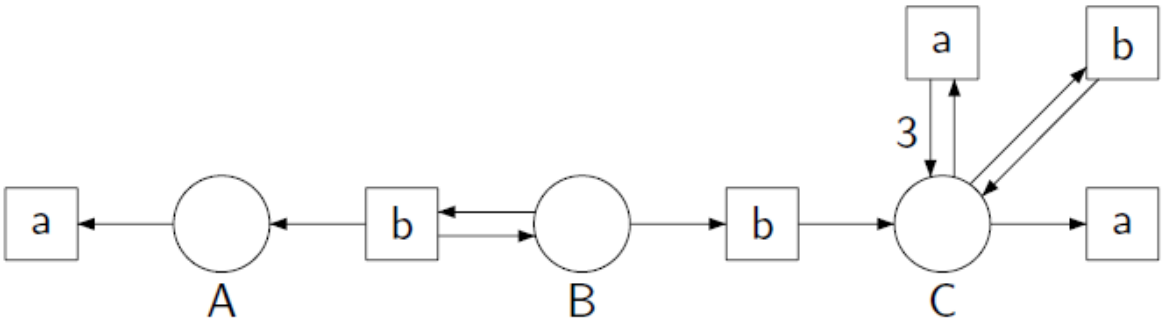
BPP systém Σ je normovaný právě tehdy, když pro každé $M \in M(\Sigma)$ platí $M \rightarrow {}^*M_\varepsilon$. Pokud je systém normovaný, můžeme vypočítat jeho normu n každého místa. Norma n místa p je nejmenší možný počet přechodů vedoucích od značení s jedním tokenem v místě p do prázdného značení. Normu místa můžeme počítat vůči libovolné podmnožině Q množiny míst BPP systému. Potom nejde o nejmenší vzdálenost k prázdnému značení, ale k nejbližšímu značení bez tokenů v množině Q. Norma systému vůči množině Q se potom dá spočítat jako $n_\Delta = \sum_{p \in Q}(n_p * M(p))$, kde n_p je norma místa vůči Q.

Každý přechod t provádí na změnu normy Δn (celé číslo, větší nebo rovno -1) na zvolené množině míst Q. Změna normy je rovna $\Delta \mathrm{n} = n_Q{}' - n_Q$, kde n_Q je norma systému pro $M\big(PRE(t)\big) = 1$ a $n_Q{}'$ je norma systému po provedení přechodu t.

2.4.1. Single final místo

O místu p řekneme, že je single final (SF) místo právě tehdy, když neexistuje přechod, který bere token z p a dává tokeny do jiného místa. Příklad viz obrázek 7([7], převzato se svolením autora, stejně jako následující obrázky ze stejného zdroje).

O místu p řekneme, že je SF-rostoucí (SF-growing) právě tehdy když jsou splněny podmínky SF místa a existuje přechod, který bere token z p a dává minimálně dva tokeny zpátky do místa p.



Obrázek 7: A je SF-místo, C je SF-rostoucí místo, B není SF-místo

2.4.2. Prime forma BPP

Každé BPP může být přetransformováno do speciální formy, zvané prime forma (prime form), kde platí $M \sim M'$ právě tehdy, když $M = M'$.

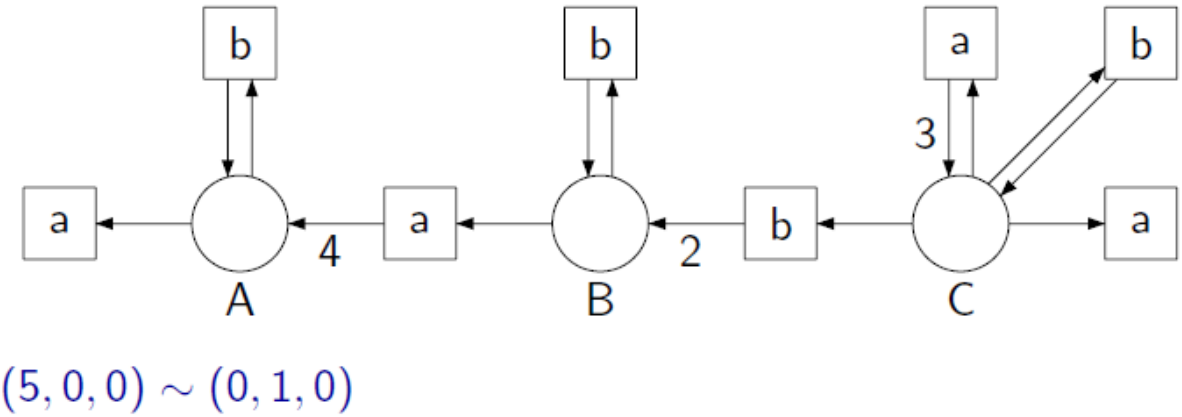
Algoritmus na převod do prime formy začíná rozkladem množiny přechodů BPP podle labelů. Pro vzniklé množiny přechodů je vypočtena norma PRE míst a změna normy všech přechodů na daných PRE místech. Množiny se rozpadávají na menší podle změny normy, tj. pokud mají přechody stejný label a stejnou změnu normy, zůstávají spolu v množině. Jsou rozkládány tak dlouho, dokud nejsou vyzkoušené všechny možnosti. Z množin přechodů jsou množiny tak, aby byl každý přechod vybrán právě jednou.[8].

Dále jsou hledány klíčové množiny. To jsou množiny, které obsahují přechody, které mají aspoň jednu změnu normy -1. Na klíčových množinách jsou hledány třídy ekvivalence relace redukce (reprezentující mezi množinami přechodů možnost vzájemného snižování možnosti uschopnění). Množiny A, B jsou ve stejné třídě tehdy, je-li $PRE(A) = PRE(B)$ a provedení nějakého přechodu z A způsobí snížení možností provedení přechodů z B.

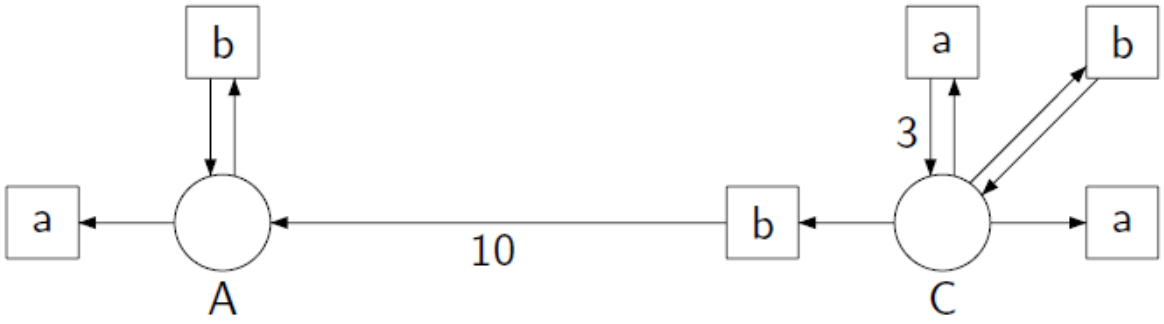
Za každou třídu ekvivalence Ekv_i je přidáváno do prime formy jedno místo p_i'. Za každý přechod t_{key} z Ekv_i je vytvořen přechod t', jež bere z $p_i' \in P'$ a dává do míst z P' odpovídajícím Ekv_i. Za každý neklíčový přechod t_{nonkey} je vytvořen přechod, jenž bere z místa $p_i' \in P'$ a dává do míst z P' odpovídajícím Ekv_i.

Množství tokenů v nově vzniklém místě odpovídá $\sum_{p \in PRE(Ekv_i)}(n_p * M(p))$, což je norma vypočtená pro místa z dané třídy vynásobené počtem tokenů v daném místě.

Příklad: Na obrázku 8 ([7]) vidíme BPP, které není v prime formě. Jelikož je BPP normovaný, může být převeden do prime formy, viz obrázek 9 ([7]).



Obrázek 8: BPP není v prime formě



Obrázek 9: BPP je v prime formě

2.4.3. Podmínky existence BPA

Pro normované BPP v prime formě existuje normované BPA, když jsou splněny následující podmínky (viz Lemma20 [1]):

1. Každé místo, které není single final, musí být omezené
2. Není dosažitelný marking, když je token v SF-rostoucím místě a ještě v nějakém dalším
3. Každé nerostoucí single final místo musí být omezené

2.5. Jednoduchá procesní algebra BPA

Jednoduchá procesní algebra (basic proces algebra – BPA, [4]) je reprezentována bezkontextovou gramatikou v Greibachově normální formě. Bezkontextovou gramatiku definujeme jako čtveřici $G = (\Pi, \Sigma, S, P)$, kde

- Π je konečná množina neterminálních symbolů (neterminálů)
- Σ je konečná množina terminálních symbolů (terminálů), přičemž $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ je počáteční (startovací) neterminál
- P je konečná množina pravidel typu $A \to \beta$
  - A je neterminál, tedy $A \in \Pi$
  - β je řetězec složený z terminálů a neterminálů, tedy $\beta \in (\Sigma \cup \Pi)^*$

Bezkontextová gramatika je v Greibachově normální formě, zkráceně v GNF, jestliže každé její pravidlo je v tvaru $X \to aY_1Y_2 \dots Y_n$ ($n \geq 0, a$ je terminál, Y_i jsou neterminály). Z BPA systému G může vzniknout LTS, které má Π jako stavy, Σ jako akce, a kde $M \xrightarrow{a} M'$ právě tehdy, když existuje pravidlo $X \to aY$ v G takové, že $X \in M$ a $M' = \Pi^*$.

Můžeme říct, že BPA systém je normovaný právě tehdy, když jsme schopni podle pravidel z počátečního neterminálu S vytvořit sekvenci, která neobsahuje neterminály.

Příklad: Gramatika v Greibachově normální formě - normovaná

$\Pi = \{A, B, C\}$
$\Sigma = \{a, b, c, z\}$
$S \to aBCB$
$B \to bC$
$B \to z$
$C \to c$

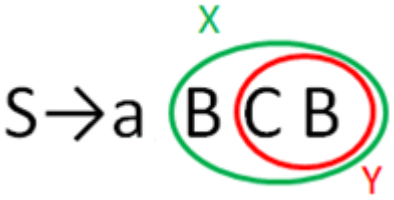
2.5.1. Transformace BPA

Transformace převádí bezkontextovou gramatiku v Greibachové normální formě na tvar $X \rightarrow aY_1Y_2$ (Y_1 a Y_2 jsou neterminály), tj. maximálně 2 neterminály na pravé straně. Transformace přináší následně úsporu času při dalších krocích porovnávání BPP a BPA, kde díky této úpravy nedochází k tak velkému procházení do hloubky.

Princip převodu si vysvětlíme pomocí následujícího příkladu:

Příklad: Transformovaná gramatika z kapitoly 2.5

1. $S \rightarrow aX$
2. $B \rightarrow bC$
3. $B \rightarrow z$
4. $C \rightarrow c$
5. $X \rightarrow bCY$
6. $X \rightarrow zY$
7. $Y \rightarrow cB$



Obrázek 10: transformace BPA

První krok transformace je zachování pravidel, které mají vpravo jeden nebo žádný neterminál (pravidla 2-4). Pravidla, která mají na pravé straně více než jeden neterminál, upravím tak, že všechny neterminály nahradím jedním novým neterminálem (pravidlo 1, nový neterminál *X*). Pro nový neterminál vytvořím tolik nových pravidel kolik jich má první neterminál zleva. Pravá strana nového pravidla vznikne provedením pravidla prvního neterminálu a přidáním zbývajícího neterminálu. V případě že neterminálů zbývá více než jeden, tak jsou opět nahrazeny novým neterminálem a proces opakuji tak dlouho, dokud nepřidám zbývající jeden neterminál.

V našem příkladu byly neterminály *BCB* nahrazeny neterminálem *X*, neterminál *B* má dvě pravidla, tudíž budou vytvářeny dvě pravidla pro neterminál *X*. Pravidlo 5 vzniklo provedením pravidla 2 za neterminál *B* a přidáním zbylých neterminálů, ty jsou ovšem dva *CB*, tak jsou nahrazeny dalším novým neterminálem *Y*. Stejným způsobem je vytvořeno pravidlo 6. Neterminál *C* má jedno pravidlo, tudíž neterminál *Y* bude mít také jedno pravidlo. Pro *C* je provedeno pravidlo 4 a je přídán zbylý neterminál, čímž vznikne pravidlo 7.

2.6. Bisimulační báze

Bisimulační bázi nazveme β_0. Formální definice z článku [1], odstavec 6.3, strana 1202:

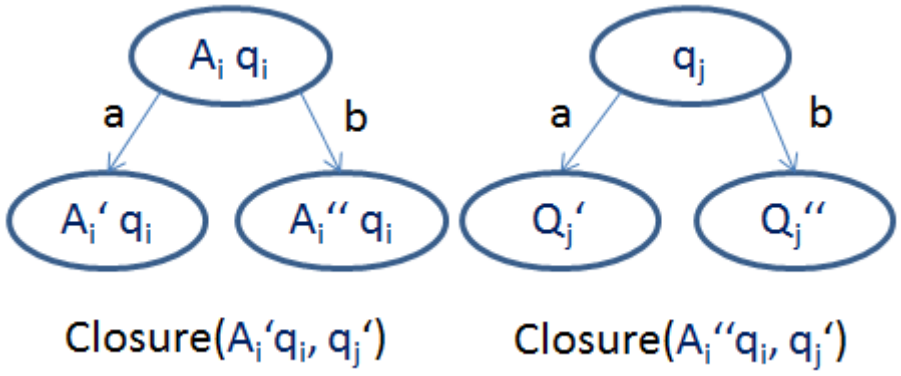
$$\beta_0 = \{(X\alpha, A | X \in V_\Sigma, \alpha \in Conf(\Sigma'), A \in F, ||X\alpha|| = ||A||\}$$
$$\cup \{(\alpha, A) | \alpha \in Conf(\Sigma'), A \in F, ||\alpha|| = ||A||\}$$

Do bisimulační báze v prvním kroku přidáme všechny elementy (dvojice a trojice míst), které splňují podmínky rovnosti norem. Pro místa z původního BPA A_i a míst ze systému BPA, které vzniklo z BPP q, I_p. I_p jsou single final místa. Způsob porovnávání norem je zobrazen v pseudokódu ve Výpis kódu 2: Pseudokód tvorby bisimulační báze.

```
forall Ai {
   Forall qj{
      If(|Ai |==|qj|)
         Add to β0 (Ai, qj);
      Forall qi {
         If(|Ai| + |qi| == |qj|)
         Add to β0 (Ai, qi, qj);
      }
      Forall Ip {
         k = |qj|-|Ai|;
         Add to β0 (Ai, I^k, qj);
      }
   }
}
```

Výpis kódu 2: Pseudokód tvorby bisimulační báze

Další krok je protřídit bisimulační bázi o elementy, které nesplňují podmínky ekvivalence. Pro každý přechod z A_i, musí být přechod z q_j, který má stejný label. Pokud mají stejný label, testuji pomocí closure, viz obrázek 11. První vstup pro metodu closure je řetězec míst vzniklých provedením přechodu z A_i a míst v q_i. Druhým vstupem je řetězec míst po přechodu z q_j. Na obrázku 11 vidíme ukázku vstupních parametrů *closure*. Pokud má každý přechod z A_iq_i aspoň jeden odpovídající přechod z q_j a obráceně, pak element zůstává v bázi, v opačném případě element z báze odstraním. Procházím tak dlouho, dokud při průchodu celou bází neodstraním žádný element.



Obrázek 11: Ukázka vstupních parametrů Closure

2.7. Closure

Formálně nadefinovaný closure najdete v článku [1], odstavec 6.3, strana 1202. Closure kontroluje, jestli platí jedna z následujících možností:

- Element je přímo obsažen v bázi.
- Pokud element je čtveřice neterminálů (RST,U) pak musí platit, že (RM,U) je v bázi a zároveň (ST,M) je v bázi
- Pokud element obsahuje neterminál vytvořený pro reprezentaci SF místa (RS, I^k), pak musí platit RS ~ I^k.

3. Modelování

3.1. Funkční požadavky

Cílem je sestrojit jednoduchý nástroj pro rozhodující ekvivalenci mezi zadaným normovaným BPP a normovaným BPA systémem. Tento nástroj musí být co nejvíce modulární, aby se do něj daly v budoucnu doplňovat algoritmy rozhodující jiné ekvivalence nebo pracující s jinými modely apod.

Vstupem tohoto nástroje bude normovaný systém BPP se značením a normovaný systém BPA se vstupním řetězcem. Systémy a značení bude možno zadávat třemi způsoby: vlastním napsáním, nahráním ze souboru a pomocí náhodného generátoru.

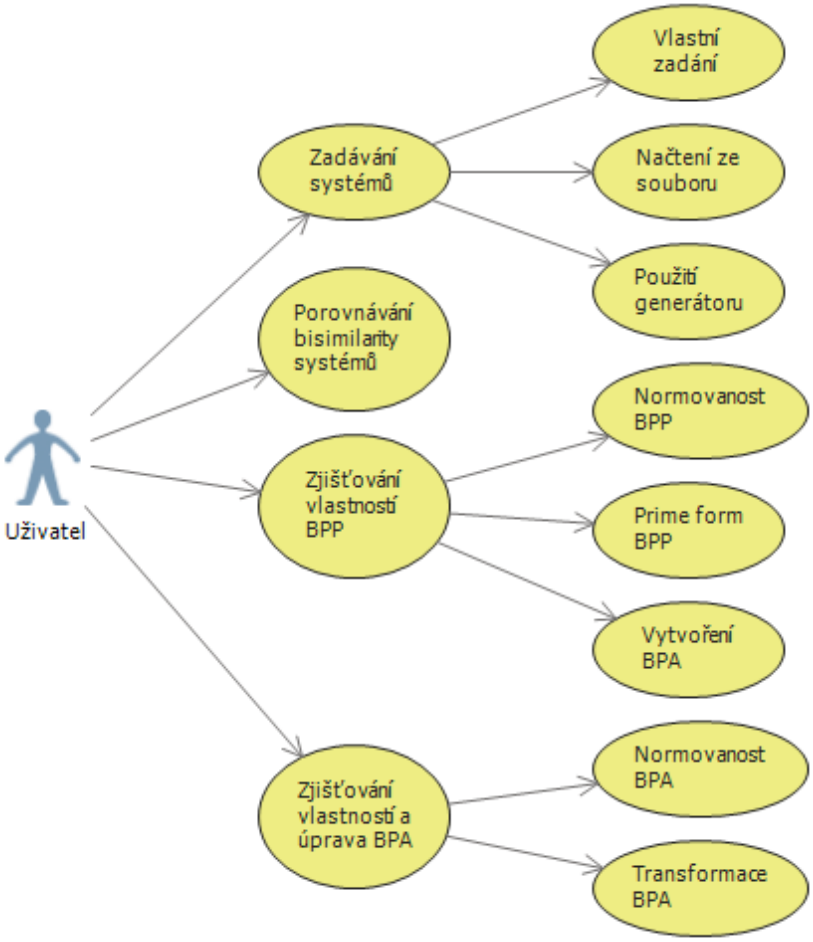
Výstupem nástroje pak bude ohodnocení, zdali jsou zadané vstupy bisimulačně ekvivalentní nebo ne. Ke zjištění použijeme algoritmus popsaný v článku [1]. Taktéž bude zaznamenávaná doba průběhu algoritmu, abychom mohli porovnat časovou náročnost s teoretickými předpoklady.

3.2. Nefunkční požadavky

Nástroj pro ověřování ekvivalence systémů je naimplementován pomocí jazyka C#, ve vývojovém prostředí Visual Studia:

- Microsoft Visual Studio 2010 Professional, Version 10.0.0.30319.1 RTMRel, na platfomě Microsoft .NET Framework Version 4.0.30319 RTMRel
- Microsoft Visual Studio Ultimate 2013, Version 12.0.21005.1 REL, na platformě Microsof .NET Framework Version 4.5.50938

3.3. Případy užití



Obrázek 12: Use Case - Diagram případů užití

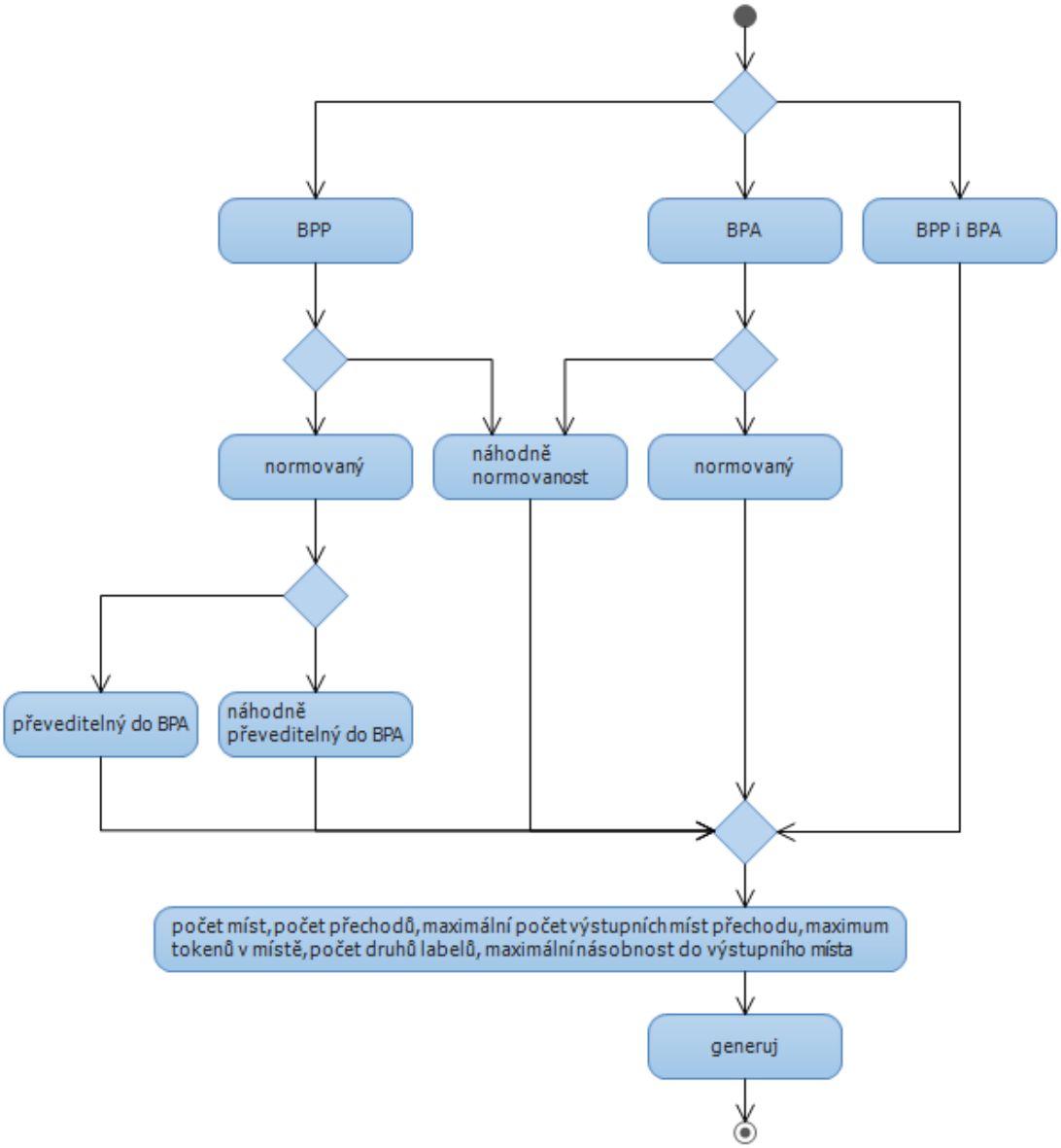
Diagram případů užití, viz obrázek 12, popisuje možnosti uživatele používat tento nástroj. Popis jednotlivých případů užití:

- Zadání systémů
- Porovnávání bisimilarity systémů
- Zjištění vlastností BPP
- Zjištění vlastností a úprava BPA

3.3.1. Zadání systémů

Uživatel zadává jednotlivé přechody vstupních systémů. U každého z nich má tři možnosti:

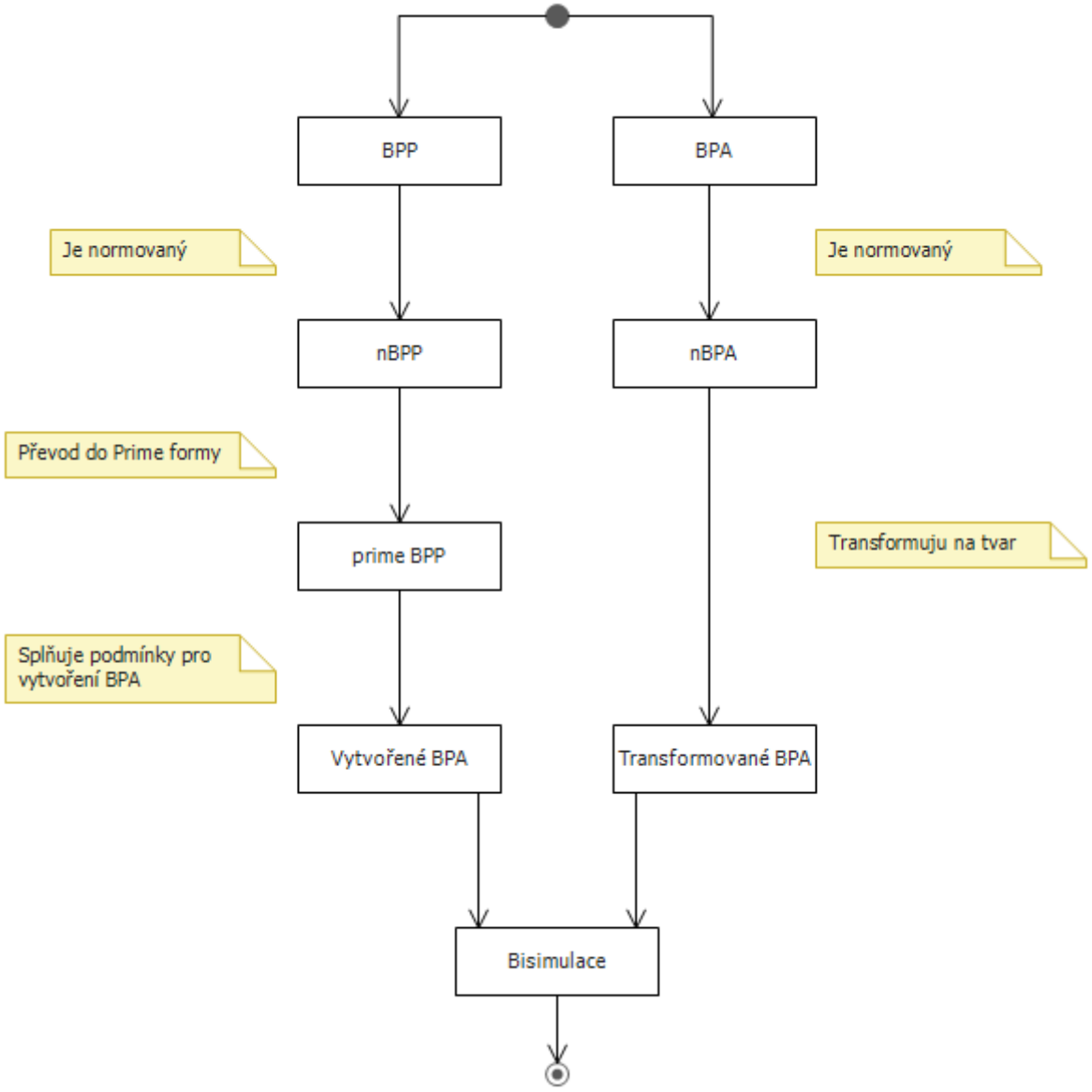
a. Vlastní zadání – Uživatel zadává jednotlivé přechody vstupních systémů.
b. Načtení ze souboru – Uživatel vybere soubor, který chce načíst jako vstupní systém nebo načíst značení případně řetězec. Uživatel dále může upravovat vstupy.
c. Použití generátoru – Uživatel nastaví parametry pro generování systémů. Možnosti generátoru a parametry jsou zobrazeny na obrázku 13.



Obrázek 13: Algoritmus generování vstupů

3.3.2. Porovnávání bisimilarity systémů

Uživatel spustí algoritmus porovnávající systémy. V první řadě je zkontrolován správný zápis systémů. Dále jsou kontrolovány vlastnosti systémů, viz obrázek 14. Nakonec se provede ověření bisimilarity. Pokud kterákoli podmínka není splněna, nejsou bisimilární.



Obrázek 14: Algoritmus porovnávání systémů

3.3.3. Zjištění vlastností BPP

Uživatel má možnost si zjistit zda je BPP normovaný, převést do prime formy a převést na BPA.

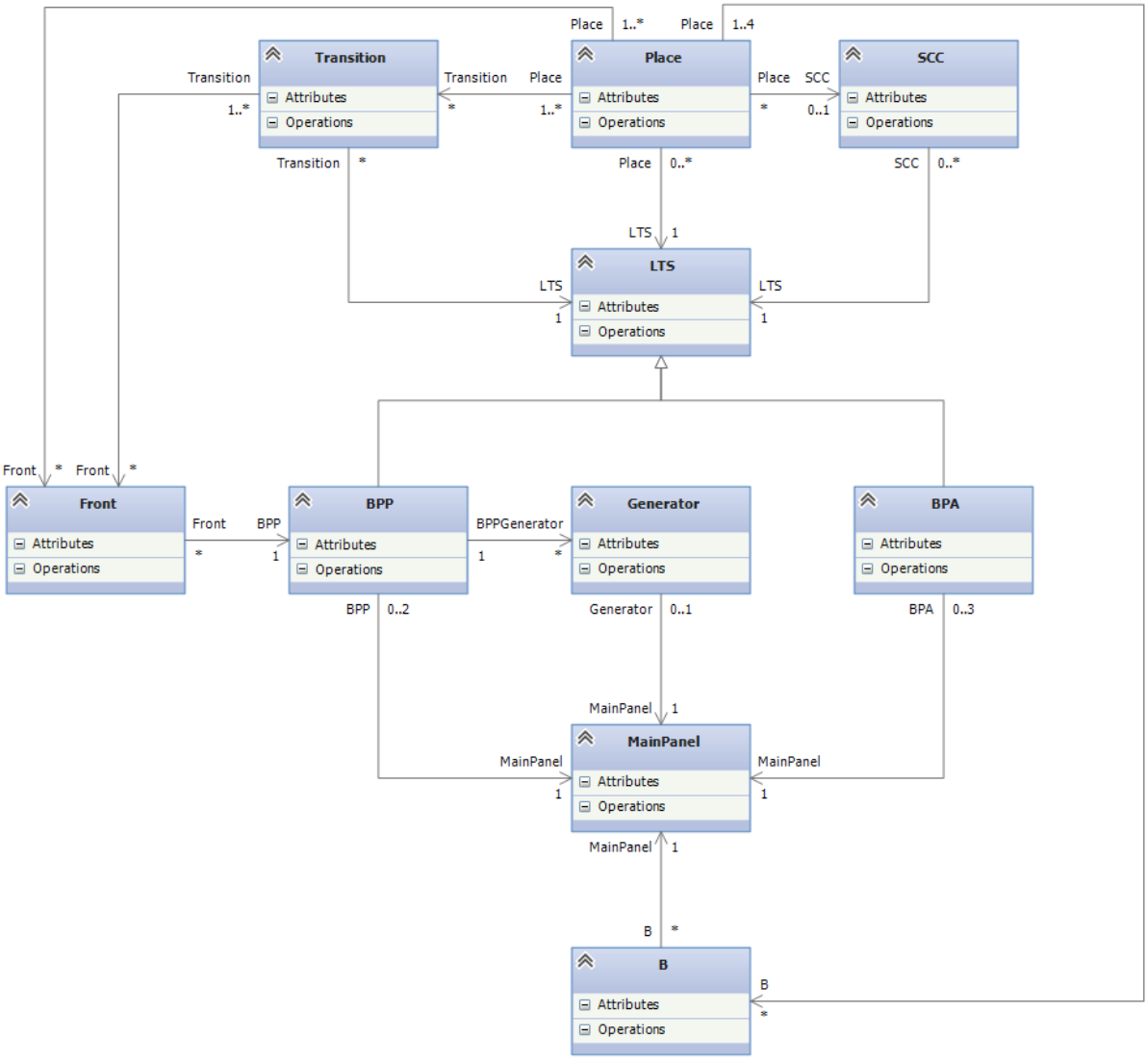
3.3.4. Zjištění vlastností a úprava BPA

Uživatel má možnost zjistit zda je BPA normované a může je také transformovat na tvar, kdy přechody mají maximálně dvě výstupní místa.

3.4. Diagram tříd

Diagram tříd popisuje statickou strukturu systému, znázorňuje datové struktury, operace u objektů a souvislosti mezi objekty. Znázorňuje datový model systému od konceptuální úrovně až po implementaci. Datové struktury zařazuje do tříd a zobrazuje vztahy těchto tříd. Diagram uvedený na obrázku 15 znázorňuje rozvržení třid v SECT.

Hlavní částí je třída *MainPanel* představující nástroj rozhodující bisimulační ekvivalenci na BPP a BPA systémech a jejich Počátečních stavů. Nástroj také umožňuje vygenerovat vstupní systémy pomocí generátoru, třída *Generator*.



Obrázek 15: Diagram tříd

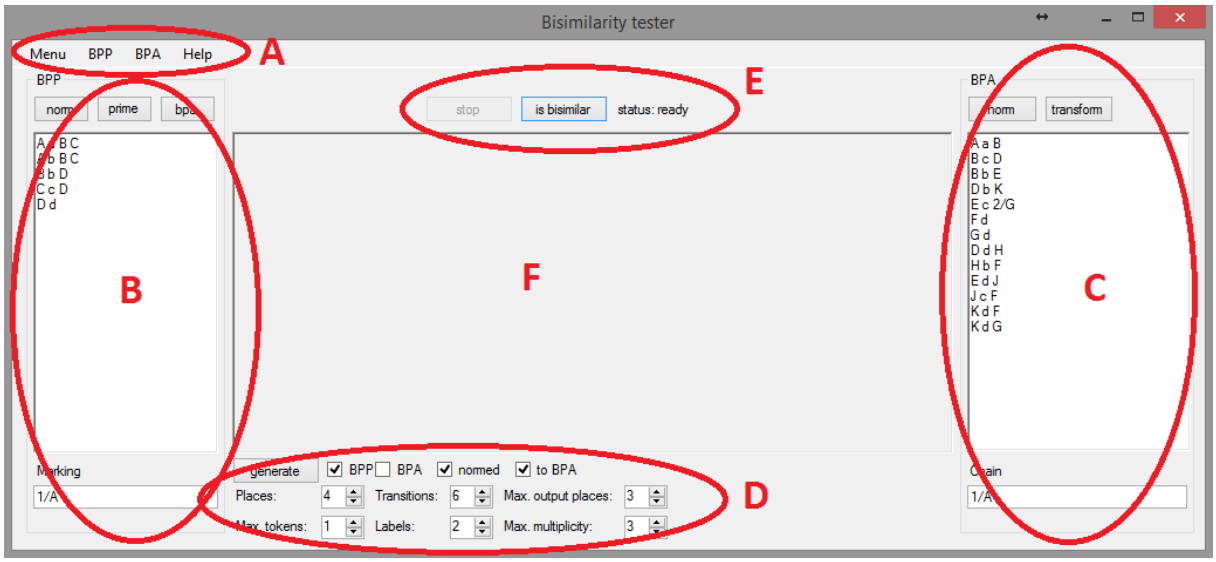
4. Implementace programu

V této kapitole se budeme zabývat popisem implementace nástroje ověřujícího bisimulační ekvivalenci na zadaných vstupech. Rozebereme si jednotlivé třídy diagramu tříd, viz obrázek 15, vysvětlíme jejich funkčnost a jaké metody implementují.

4.1. Vzhled nástroje

Na obrázku 16 je zobrazen vzhled hlavního panelu, ten se skládá z:

- Menu – A – Obsahuje základní a rozšířené funkce nástroje. Umožňuje otevření a ukládání jednotlivých systémů a jejich vstupních hodnot. Také je možnost nápovědy.
- Panel pro BPP – B – Zobrazuje vstupní systém BPP a jeho marking. Dále je možné testovat dílčí výpočty.
- Panel pro BPA – C - Zobrazuje vstupní systém BPA a jeho řetězec. Dále je možné testovat dílčí výpočty
- Generátor – D – Zde má uživatel možnost vygenerovat nový BPP nebo BPA, případně oba.
- Hlavní oblast – E – Tlačítko pro přerušení jakékoli operace, hlavní výpočet bisimilarity, aktuální stav nástroje, kterou zrovna provádí.
- Výstupní oblast – F – Tady uživatel vidí výsledky zjištěné jak přes dílčí výpočty, tak i konečné rozhodnutí zda jsou systémy bisimulačně ekvivalentní.



Obrázek 16: Vzhled nástroje

4.2. Reprezentace nástroje

Třída *MainPanel* reprezentuje tento nástroj, tato třída pro svou funkcionalitu dále využívá třídy *BPP* (4.2.3) a *BPA* (4.2.4), které dědí ze třídy *LTS* (4.2.2). Třída *Generátor* (4.2.8) je další třída, která je využívána.

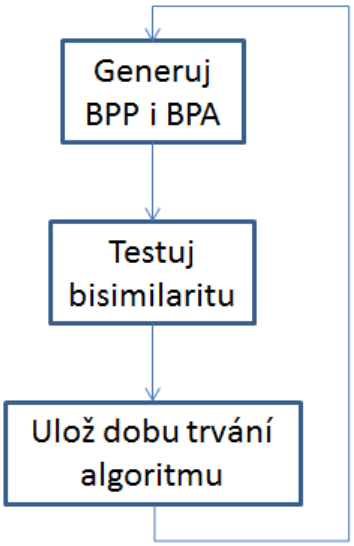
4.2.1. Třída MainPanel

Reprezentuje nástroj, tudíž i ovládání rozhraní a konečný výpočet bisimulace. Taktéž veškerou činnost tlačítek (*norm, prime, bpa, prime, stop, is bisimilar, norm ,transform, generate*)a horní lišty nástroje (*Menu, BPP, BPA, HELP*). Uživatel má možnost sledovat aktuální činnost nástroje při déle trvající operaci (*status*). Pomocné mezi výpočty a konečný výsledek jsou zobrazen v prostřední části (*output*). Uživatel může využít generátoru, kterému nastaví vstupní parametry (zaškrtávací políčka *BPP, BPA, normed, to BPA*. Numerické číselníky *Places, Transitions Max. output places, Max. tokens, Labels, Max. multiplicity*).

Nejdůležitější metodou je *Calculation* se vstupním parametrem string, který určuje kdy má metoda skončit. Tato metoda vyhodnocuje požadavky od uživatele. Metoda je vždy spuštěna na pozadí v druhém vlákně, to umožňuje uživateli metodu kdykoli přerušit. Pro rozhodnutí bisimulace je využívána třída *B*, která reprezentuje elementy báze, ta má implementované dvě metody:

- *Selector* – Metoda slouží k protřídění báze. Ve chvíli kdy máme bázi kompletní, je v ní plno elementů, které do báze nepatří. Teoretický základ je popsaný v kapitole 2.6.
- *Closure* – Metodu využijeme několikrát, slouží k ověření elementů vzniklých po provedení přechodu. Také je klíčová v posledním kroku zjišťování bisimilarity, kdy je testováno, jestli v bázi je element vzniklý ze vstupního řetězce BPA a vstupního místa vytvořeného BPA z BPP, což je místo odpovídající vstupnímu markingu BPP.

Tlačítko *Loop* není defaultně zobrazeno, uživatel má možnost si ho zobrazit. Slouží ke spuštění nekonečné smyčky třech metod: *Generate, Calculation, SaveResults.* Zobrazeno na obrázku 17. Jediný způsob jak smyčku přerušit je použít tlačítko *Stop*. Výsledky jsou ukládány do externího souboru *TimeResults.txt*.



Obrázek 17: Nekonečná smyčka

4.2.2. Třída LTS

Třída *LTS* představuje společné vlastnosti a funkce tříd *BPP* (4.2.3) a *BPA* (4.2.4). Její funkce slouží k dalšímu nastavení všech vlastností *LTS* po načtení z textové podoby a jejich výpis pro uživatele. Třída využívá dalších tříd: *Place* (4.2.5), *Transition* (4.2.6), *SCC*(4.2.7). Na třídním diagramu systémů, viz obrázek 20, můžeme vidět všechny atributy a metody tříd reprezentující systémy. Metody můžeme rozdělit na:

- Metody pro zjištění silně souvislých komponent: *CreateSCCTarjanAlg, StrongConnect, SeTokensSCC, SetDependenciesSCC*
- Metody pro zjištění vlastností komponent, tudíž i míst v nich: *SetReachebleForMarkingSCC, SetBoundedSCC, SetReachebleForMarkingSCCRekurze, SetBoundedSCCRekurze, SetSCCSingleFinal, SetNormOfPlaces, SetNormPlaceNotCount, CountChangeNorm*
- Metody pro zjištění zda je LTS normovaný: *IsNormed*
- Pomocné metody: *CallSets, PrintPlaces, PrintTransitions, RenameTransitions, RenamePlaces*

Metoda *CallSets*, viz Výpis kódu 3: Ukázka metody CallSets, ukazuje posloupnost spouštění jednotlivých metod pro získání všech informací při tvorbě nového LTS.

```
public void CallSets()
{
   SetTransitions();
   RenameTransitions();
   CreateSCCTarjanAlg();
   SetTokensSCC();
   SetDependenciesSCC();

   SetReachebleForMarkingSCC();
   SetBoundedSCC();
   SetSCCSingleFinal();
   SetNormOfPlaces(this.places);
   CountChangeNorm();

   IsNormed();
}
```

Výpis kódu 3: Ukázka metody CallSets

4.2.3. Třída BPP

Představuje vstupní BPP systém. Jeho metody slouží k načtení informací z textové podoby. Vytvoření BPP v prime form k čemuž využívá pomocnou třídu *Front* pro rozklady podle značení a změny normy. Ověřuje podmínky existence BPA a jeho vytvoření. Metody:

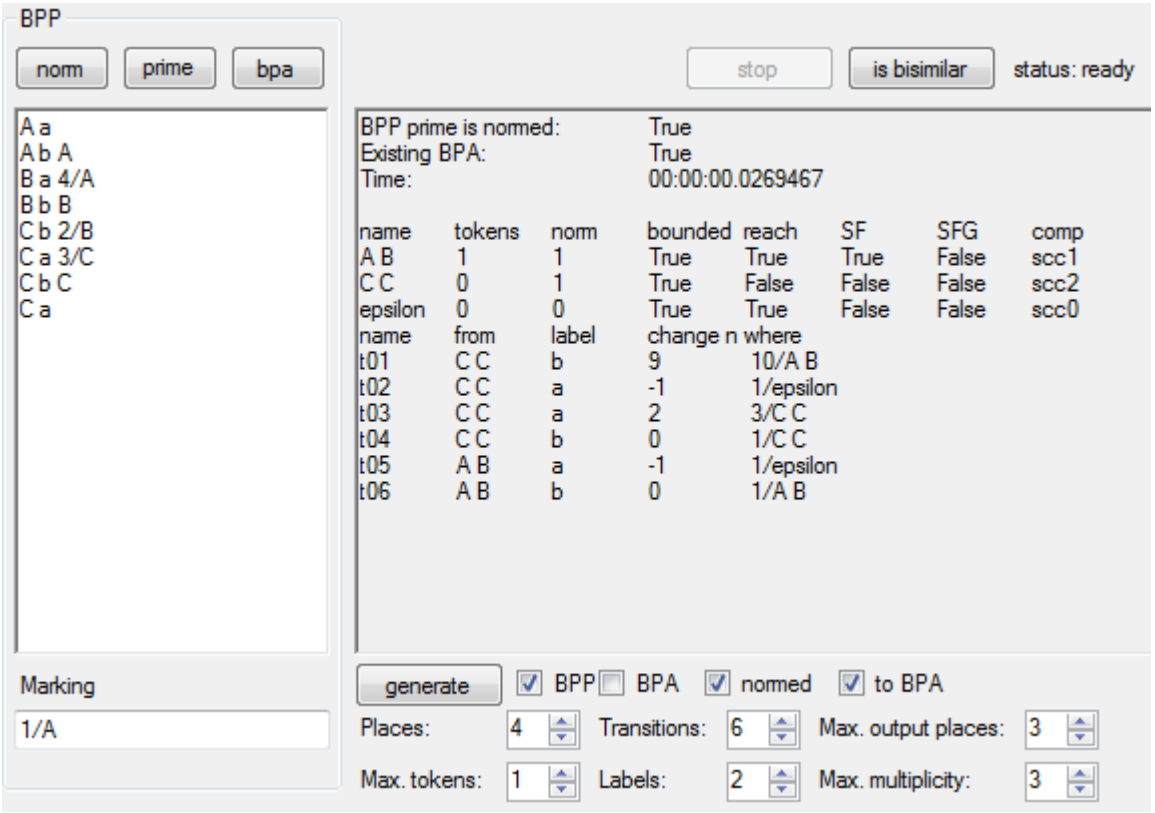
4.2.3.1. Vytvoření BPP ze vstupu

Použité metody: *SetBPP, SetMarking.* Následně je volána metoda *CallSets* (viz. Výpis kódu 3).

4.2.3.2. Pro tvorbu prime form

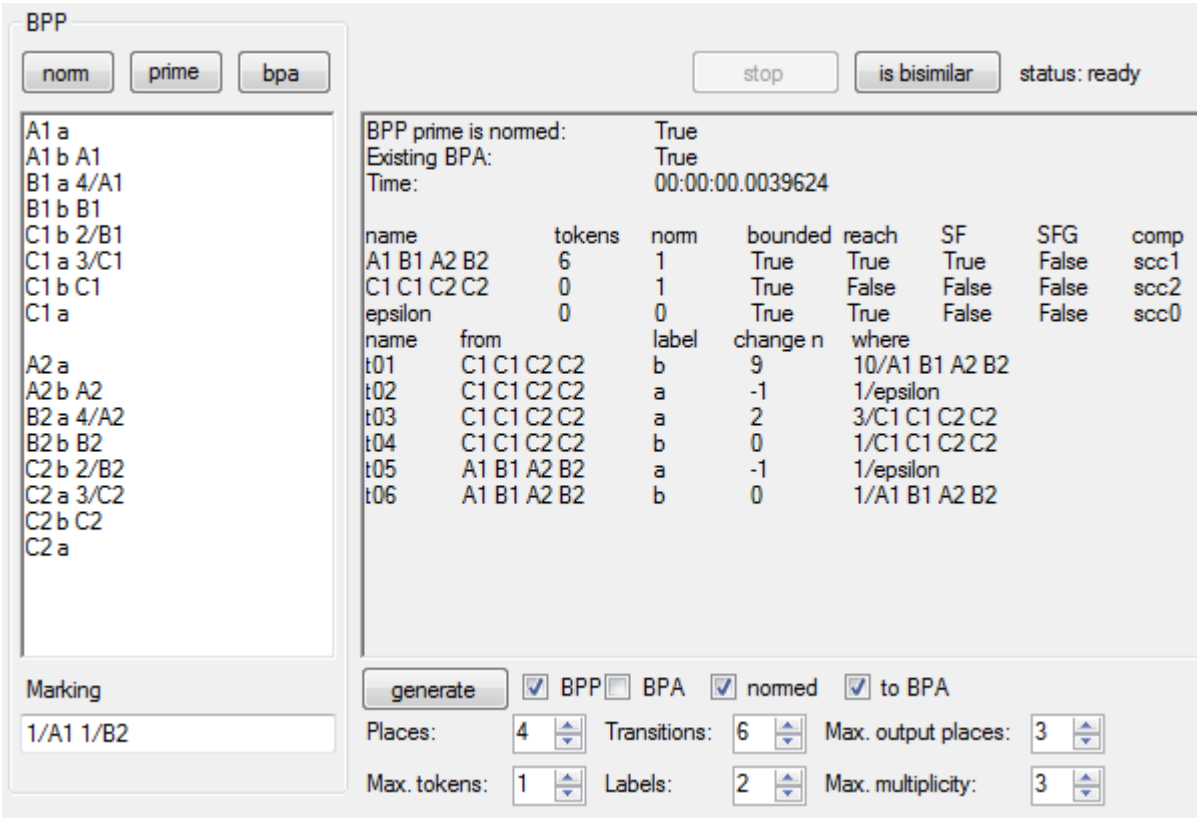
Použité metody: *CreatePrimeForm, GetPrimePlace, Split, SplitByLabel, SplitByChangeNorm, CountChangeNorm, ResetChangeNor.*

Zapíšeme-li příklad z obrázku 8, jako vstupní BPP. Převedeme do prime form, dostaneme BPP odpovídající obrázku 9. Výstup z programu vidíme na obrázku 18.



Obrázek 18: Výstup z programu, převod do prime form 1

Pokud použijeme jako vstupní BPP opět systém z obrázku 8, ale tentokrát tento systém vložíme dvakrát s přejmenovanými místy. Po převedení do prime form bude opět odpovídat systému na obrázku 9. Výstup z programu vidíme na obrázku 19.



Obrázek 19: Výstup z programu, převod do prime form 2

4.2.3.3. Ověření existence BPA

Použité metody: *ConditionsExistingBPA, ExistingBPA2, ExistingBPA2Rekurze*.

Metody ověřují podmínky z kapitoly 2.4.3. Podmínky existence BPA. Nejprve jsou ověřeny podmínky 1 a 3 v metodě *ConditionsExistingBPA*, které jsou méně náročné. Následně je z této metody volána funkce *ExistingBPA2*, která zjišťuje platnost druhé podmínky, také využívá rekurzivní metody *ExistingBPA2Rekurze*.

4.2.3.4. Tvorba BPA

Použité metody: *CreatBPA, CreatBPARekurze, OneMark, GetNameByTokens*.

BPA se začíná vytvářet z markingu BPP v prime form, což je první místo nově vzniklého BPA. Toto obstarává metoda *CreatBPA*. Následně se provádějí všechny možné přechody, čímž vznikají nové markingy, odpovídající místům v BPA. Průchod všemi markingy provádí CreatBPARekurze. V případě, že místo je SF-rostoucí, netvoří se další markingy, ale je nahrazeno *-1*.

4.2.4. Třída BPA

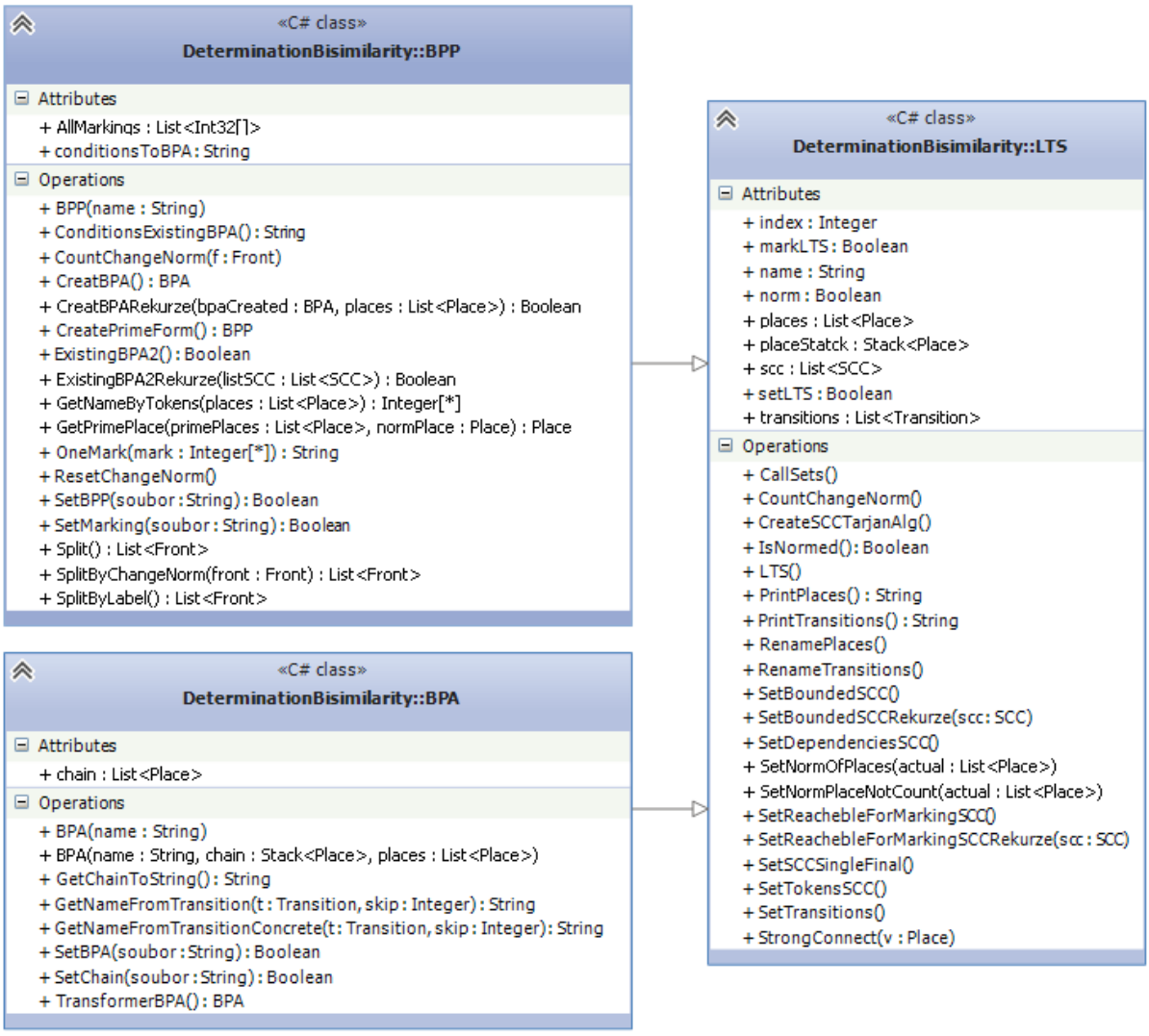
Představuje vstupní BPA systém. Jeho metody slouží k načtení informací z textové podoby, převedení BPA na takový tvar, že na pravé straně budou maximálně dva neterminály. Metody:

4.2.4.1. Vytvoření BPA ze vstupu

SetBPA, SetChain. Následně je volána metoda *CallSets* (Výpis kódu 3: Ukázka metody CallSets)

4.2.4.2. Transformování BPA

TransformerBPA, GetNameFromTransition, GetNameFromTransitionConcrete



Obrázek 20: Diagram tříd LTS, BPP, BPA

4.2.5. Třída *Place*

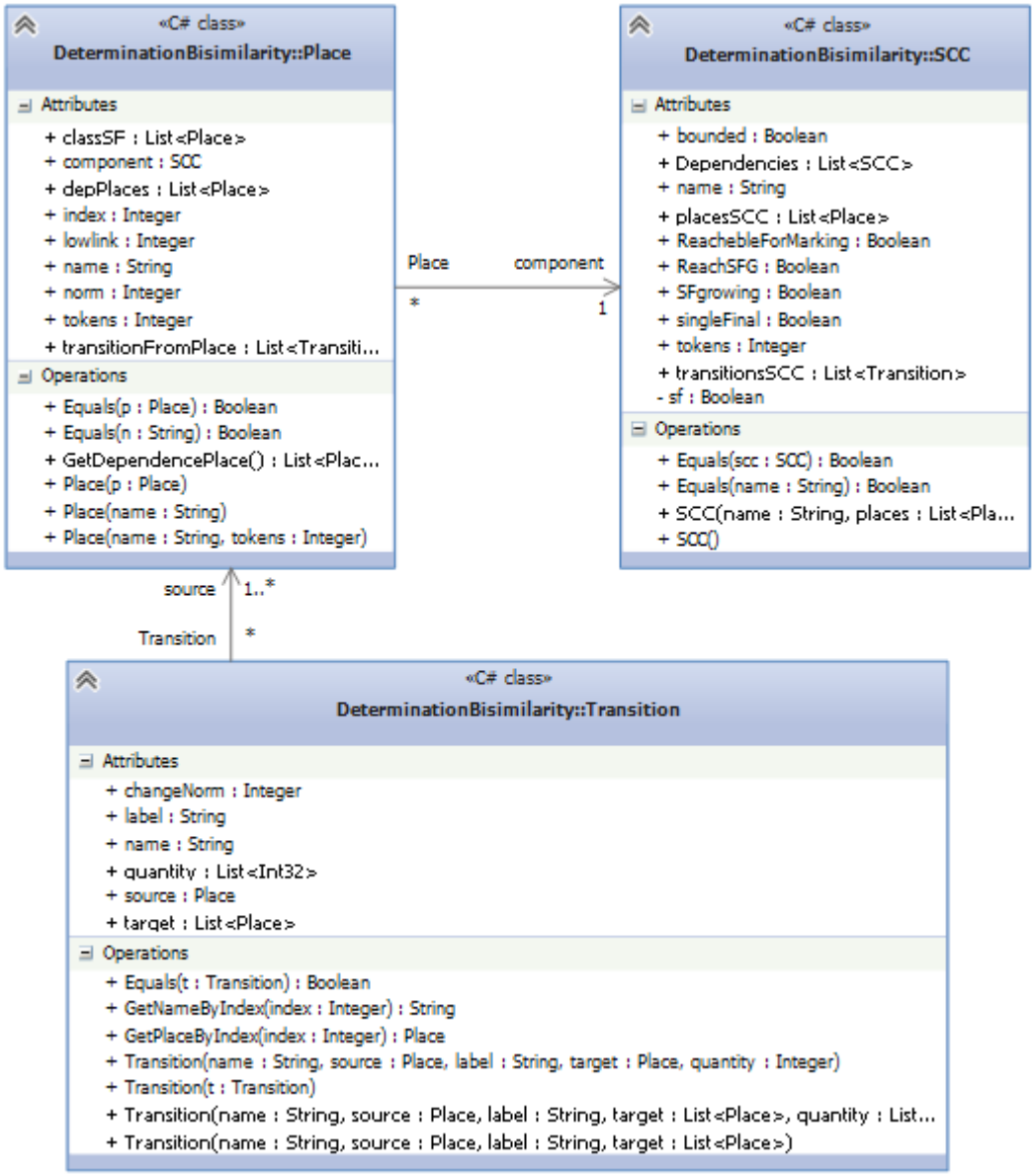
Reprezentuje místo z konečné množiny $P \in \Sigma$, tedy místo z konečné množiny míst LTS systému. Jak je vidět v třídním diagramu na obrázku 21, každé místo má označení *name*, dále normu *norm*, počet tokenů *tokens*, komponentu *component* (objekt typu *SCC*), přechody jdoucí z místa *transitionFromPlace* (objekt typu *List<Transition>*), závislé místa *depPlaces*(objekt typu *List<Place>*).

4.2.6. Třída *Transition*

Reprezentuje přechod z konečné množiny $Tr \in \Sigma$, tedy přechod z konečné množiny přechodů LTS systému. Jak je vidět v třídním diagramu na obrázku 21, skládá se z označení přechodu *name*, vstupního místa *source*(objekt typu *Place*), ohodnocení přechodu *label*, výstupní místa *target* (objekt typu *List<Place>*) a násobnost *quantity* (objekt typu *List<int>*), změna normy *changeNorm*.

4.2.7. Třída *SCC*

Třída reprezentuje místa, která tvoří silně souvislou komponentu (kapitola 2.2.4) v LTS. Jak je vidět v třídním diagramu na obrázku 21, skládá se z listu míst *placesSCC* (objekt typu *List<Place>*) a list přechodů *transitionsSCC* (object typu *List<Transition>*). Vlastnosti komponenty, které jsou shodné s vlastnosti jednotlivých míst: omezenost *bounded*, dosažitelnost *ReachebleForMarking,* zda je single final *sf*, zda je single final rostoucí *SFgrowing*.



Obrázek 21: Třídní diagram Place, Tranisiton, SCC

4.2.8. Třída *Generator*

Třída *Generator* slouží k vygenerování BPP a BPA. Konstruktor generátoru obsahuje všechny vlastnosti pro nastavení generátoru, číselné hodnoty: počet míst *nPlace,* počet přechodů *nTransition*, maximum tokenů v místě *maxToken*, počet druhů labelů *nLabel*, maximální počet výstupních míst přechodu *nMaxTarget*, maximální násobnost do výstupního místa přechodu nMaxQuantity. Hodnoty True/False, které slouží k definování, co uživatel chce generovat: generování BPA *checkBPA*, generování BPP *checkBPP*, normovaný systém *checkNormed*, BPP převoditelné do BPA *checkTo*.

Pro generování pouze systému BPA slouží metoda *GenerateBPA.* Pro generování systému BPP slouží metoda GenerateBPPorBoth, v případě, že jsou generovány oba systémy, je použita tatáž metoda. BPA pak vznikne z vygenerovaného BPP tak, že je převeden na BPA.

Samotné generování probíhá následovně: nejprve se vytvoří jedno místo, k němu jeden přechod a testují se podmínky, konkrétně podmínka normovanosti a podmínka převoditelnosti, pokud je uživatel zadal. Pokud jsou podmínky splněny, pokračuje se dál dalším místem a dalším přechodem. Pokud nejsou splněny, přechod je smazán a generuje se nový přechod. Ve chvíli, kdy jsou vygenerována všechna místa a ke každému právě jeden přechod, tak se dogenerují přechody navíc. Po vytvoření každého přechodu jsou testovány podmínky, pokud nejsou splněny, je přechod smazán. Při generování BPA je ještě třeba vygenerovat vstupní řetězec, ten je generován z tokenů, které jsou v místech generovány.

5. Vstupy do programu

V této kapitole se budeme zabývat zadáváním vstupních systémů BPP a BPA. Pro tento účel existují celkem tři možnosti, tedy:

- vlastní zadání
- načtení ze souboru
- vygenerování náhodného normovaného vstupu

5.1. Zadání vstupů z klávesnice

5.1.1. Systém

Zahrnuje vytvoření systémů pomocí samotného napsání jednotlivých přechodů, ale také uživateli umožňuje rozšířit již existující vstup (tj. systém vygenerovaný nebo otevřený ze souboru) o další přechody. Systémy zadáváme jako přechody. Nejprve napíšeme vstupní místo, ohodnocení přechodu (tj. akce vedoucí ze vstupního místa) a množinu výstupních míst, kterou tvoří jednotlivá výstupní místa s přiřazenou násobností hrany. K reprezentaci takových dat jsem zvolil následující podmínky a doporučení:

- Jeden přechod odpovídá jednomu řádku.
- Vstupní místo musí být tvořeno libovolně dlouhým řetězcem znaků i čísel, který neobsahuje mezery. Doporučení: použít jedno písmeno velké abecedy.
- Každá akce (ohodnocení přechodu) je omezena taktéž na jeden souvislý řetězec znaků. Doporučení: jeden znak malé abecedy.
- Výstupní místo omezují stejné podmínky jako místo vstupní. Pouze pro případ, kdy bude přechod vyprazdňující, volíme jeho výstupním místem prázdný řetězec (viz příklad2), ten je pak nahrazen řeckým písmenem epsilon (ε). Další výstupní místa píši s mezerou stejným způsobem (viz příklad 3). Pokud je uvedeno pouze výstupní místo, pak je násobnost brána automaticky 1 (viz příklad 4).
- Násobnost hrany u výstupních míst poté reprezentuje čísla od 1 do námi zvolené libovolné hodnoty.
- Jednotlivé části jsou od sebe odděleny mezerou, kromě násobnosti u výstupního místa, to je psáno následovně: NASOBNOST/VYSTUPNI_MISTO (viz příklad 3).

Příklad 1: Vstup akce 2/vystup
Příklad 2: B b
Příklad 3: A f 2/B 3/C 1/D
Příklad 4: A a D

5.1.2. Vstup systému

Pro zápis vstupu do systému platí stejné pravidlo, jako pro výstupní místa přechodů. Tj. NASOBNOST/MISTO, další místa jsou od sebe oddělena mezerou. Lze také psát pouze MISTO, automaticky bude přiřazena násobnost jedna.

Příklad 1: 2/B 3/C 1/D
Příklad 2: D B B C

5.2. Načtení ze souboru

Načítání systémů nebo jejich vstupů se provádí přes menu příslušného systému (např: BPA → Load Chain). Výstup se zobrazí v příslušném textBoxu. Pravidla pro zapisování jsou shodná s pravidly se zadáváním z klávesnice, viz kapitola 5.1.

Příklad: systému BPP uloženého v textovém souboru

A b C
B b
B a 3/B
C a
C c 2/A
D a 1/A 1/B

5.3. Generátor

Uživatel má možnost generovat BPP, BPA, nebo oba najednou a jejich vstupy s určitými vlastnostmi. Pro správnou funkčnost generátoru je třeba nastavit následující parametry:

- Počet
    - Places – Počet míst v LTS
    - Transitions – Počet přechodů v LTS
    - Max. output places – Maximální počet výstupních míst přechodu
    - Max tokens – Maximální množství tokenů v místě pro BPP, pro BPA kolikrát maximálně může být místo ve vstupním řetězci
    - Labels – Počet druhů labelů, písmenka malé abecedy, max. 26
    - Max multiplicity – Maximální násobnost přechodu do výstupního místa
- True/False hodnota
    - BPP – pokud chci generovat systém BPP
    - BPA – pokud chci generovat systém BPA
    - Normed – pokud chci, aby systém byl normovaný. Pokud má být systém normovaný, musí být přechodů minimálně tolik, kolik je míst.
    - To BPA – pokud chci, aby BPP bylo převeditelné do BPA. Možné jen v případě, že generuji BPP nebo oba systémy, viz níž.

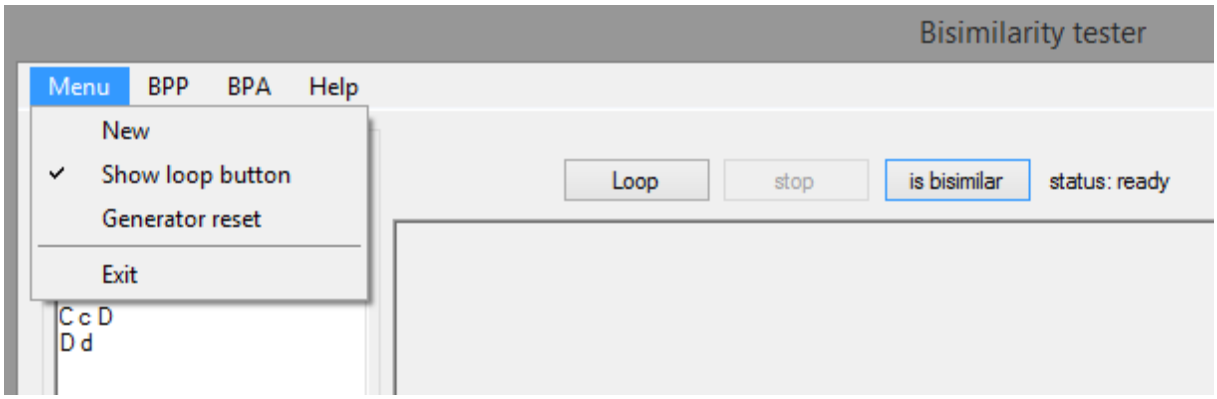
Vytváří-li generátor normovaný nebo převeditelný do BPA systém, vytvoří jedno místo a z něho jeden přechod. Pak otestuje, zda je normovaný (převoditelný do BPA). Takto se postupuje pro všechna místa. V dalším kroku vytvoří zbývající přechody, po přidání každého přechodu je testováno, zda jsou parametry splněny.

V případě, kdy jsou generovány oba systémy najednou, je automaticky požadováno, aby oba systémy byly normované a systém BPP převeditelný do BPA. Systém BPA pak bude vytvořen z vygenerovaného BPP systému.

Pokud se generátoru nepodaří vygenerovat požadovaný systém(y), je možné pro stejné parametry spustit generování znovu nebo parametry upravit.

5.3.1. Tlačítko Loop

Uživatel má možnost využít nekonečnou smyčku. Aby ji mohl využít, musí nejprve zobrazit tlačítko Loop (*Menu → Show loop button*). Poté nastaví numerické hodnoty generátoru podle vlastní úvahy, zaškrtávací políčka budou automaticky zaškrtnuta. Následným stisknutím tlačítka *Loop* dojde ke spuštění nekonečné smyčky, začnou se generovat systémy BPP a BPA, zjišťovat jejich bisimilaritu a ukládat časový záznam algoritmu do externího souboru.



Obrázek 22: Tlačítko Loop

6. Časová složitost algoritmu

V této kapitole nás bude zajímat celkový čas výpočtu při ověřování bisimulační ekvivalence na zadaných vstupech pomocí porovnávacího algoritmu. Tyto vstupy budou mít různou velikost. Algoritmus bude prováděn na 3 rozdílných zařízeních. Získané výsledky budou porovnány a vyhodnoceny.

6.1. Testovací zařízení

A. Notebook s operačním systémem Windows 8.1 Pro. Procesor Intel Core i7-3632QM CPU 2.20 GHz. Paměť RAM 8192. 64 bitový OS. Dále jako notebook A.
B. Notebook s operačním systémem Microsoft Windows 7 Professional, Service Pack 1. Procesor Intel Pentium CPU B970 2,3 GHz. Paměť RAM 4096 MB. 32 bitový OS. Dále jako notebook B.
C. Server s operačním systémem Microsof Windows Server Standart 2007, Service Pack2. Model IBM Systém x 3200 M2, Procesor Intel Xeon CPU X3330 2,66 GHz. Paměť RAM 4096MB. 64 bitový OS

6.2. Testované vstupy

Testované vstupy jsou vytvářené pomocí generátoru. Generátor je nastaven tak, aby vytvářel BPP i BPA, čímž vznikají systémy, které jsou bisimulačně ekvivalentní. Generátor je nastaven na počet míst 4, počet přechodů 6, maximální počet výstupních míst 3, maximum tokenů v místě 1, počet druhů labelů 2, maximální násobnost do výstupního místa 3.

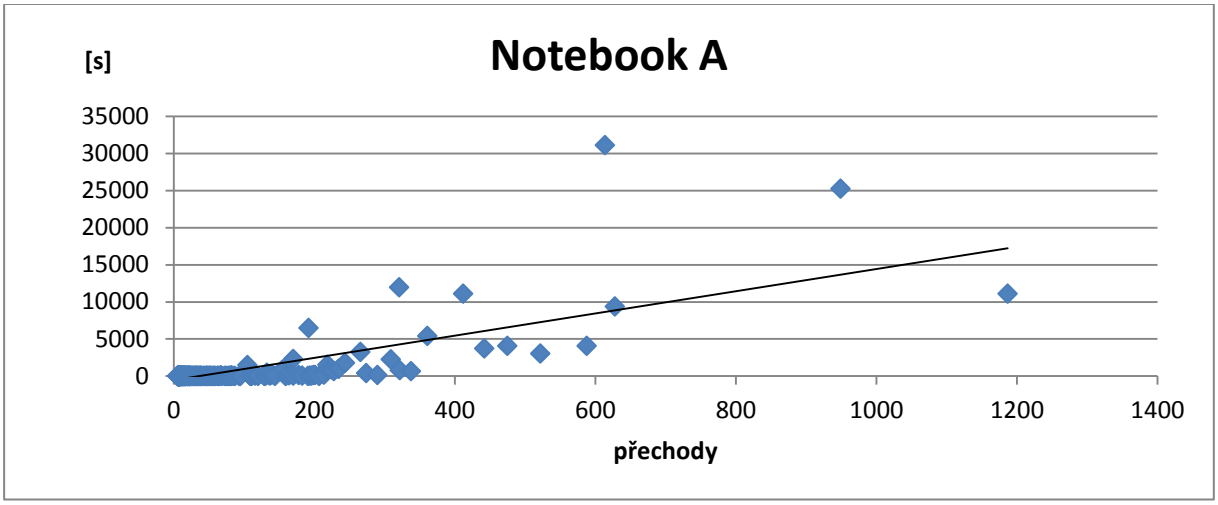
6.3. Výsledky na jednotlivých zařízeních

Za vypovídající data jsem zvolil počet vstupních přechodů obou systémů v závislosti na čase. Všechny testované systémy jsou bisimulačně ekvivalentní. Časově nejnáročnější metoda je *Selector*, která porovnává všechny možné elementy báze, viz kapitola 2.7.

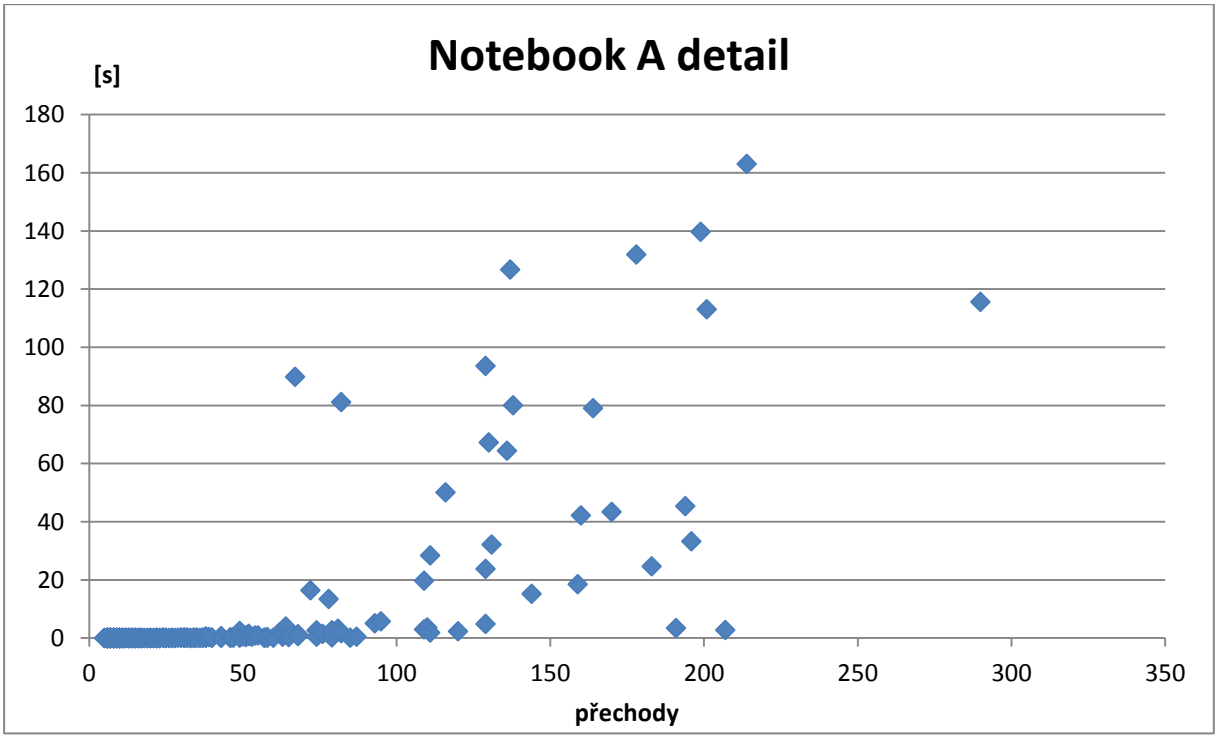
U některých testovaných dvojic je čas výrazně vyšší než čas pro dvojici s přibližně stejným počtem přechodů.

6.3.1. Notebook A

Na notebooku A bylo otestováno 305 dvojic systémů BPP a BPA, které byly bisimulačně ekvivalentní. Na grafu 1 vidíme všechny dosažené výsledky. Dvojice, které mají dohromady do 200 přechodů, jsou vyhodnocené za cca 1 minutu, viz graf 2.



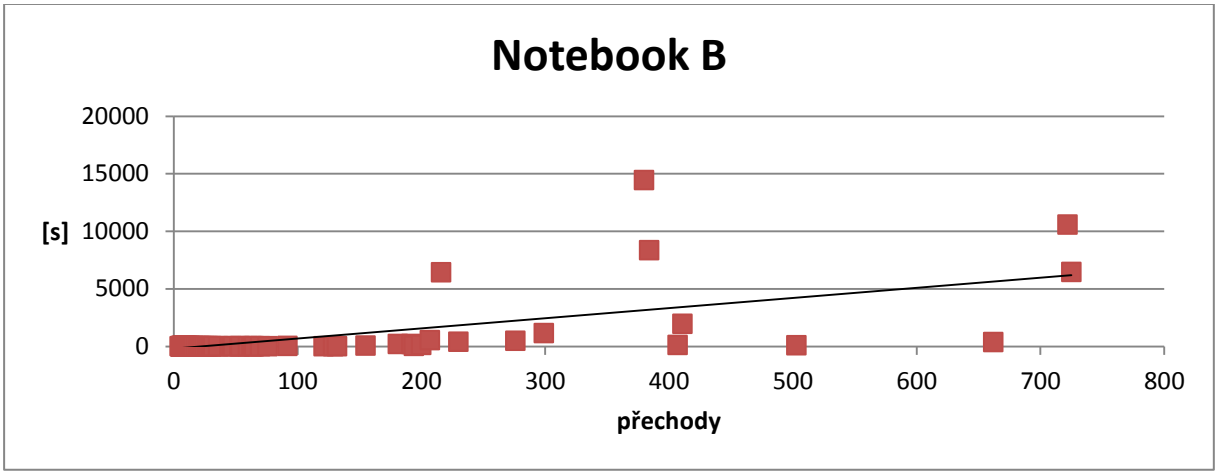
Graf 1: Notebook A celkový přehled



Graf 2: Notebook A detail

6.3.2. Notebook B

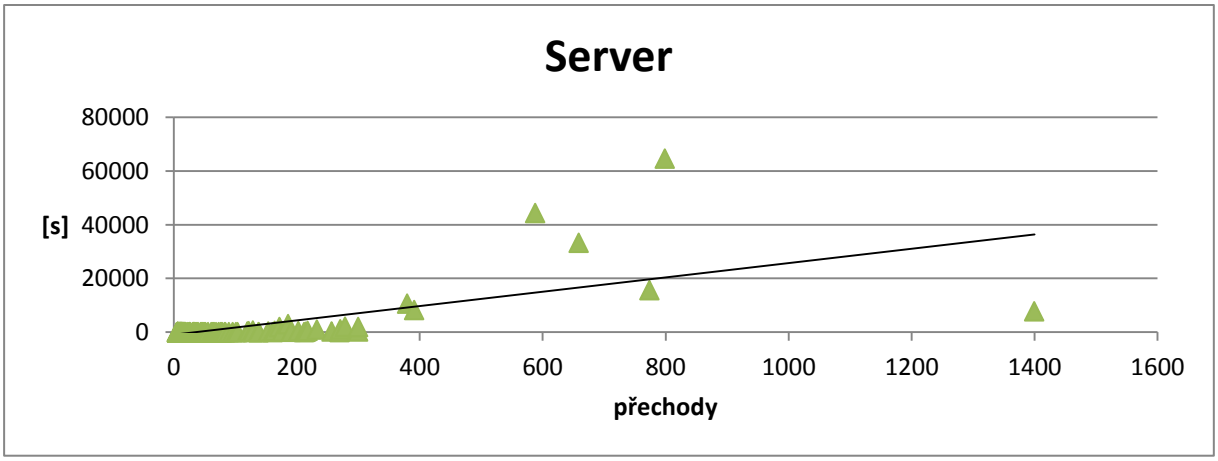
Na notebooku B bylo testováno 106 dvojic. Na notebooku B byla testována dvojice, která čítala dohromady 375657 přechodů. Doba trvání algoritmu byla necelých 8hodin. Tato dvojice není zahrnuta v grafu 3.



Graf 3: Notebook B celkový přehled

6.3.3. Server

Na posledním testovacím zařízení bylo testováno 147 dvojic. Během testování byli k serveru připojeni další uživatelé a byl využíván k dalším procesům, což je pravděpodobně příčina vyšších výsledků časů u dvojic s vyšším počtem přechodů. Na grafu 4 vidíme celkový přehled.



Graf 4: Server celkový přehled

6.4. Vyhodnocení

Jelikož všechny systémy jsou bisimulačně ekvivalentní, nástroj musí porovnat všechny možnosti, jedná se tak o nejhorší možný průchod algoritmu.

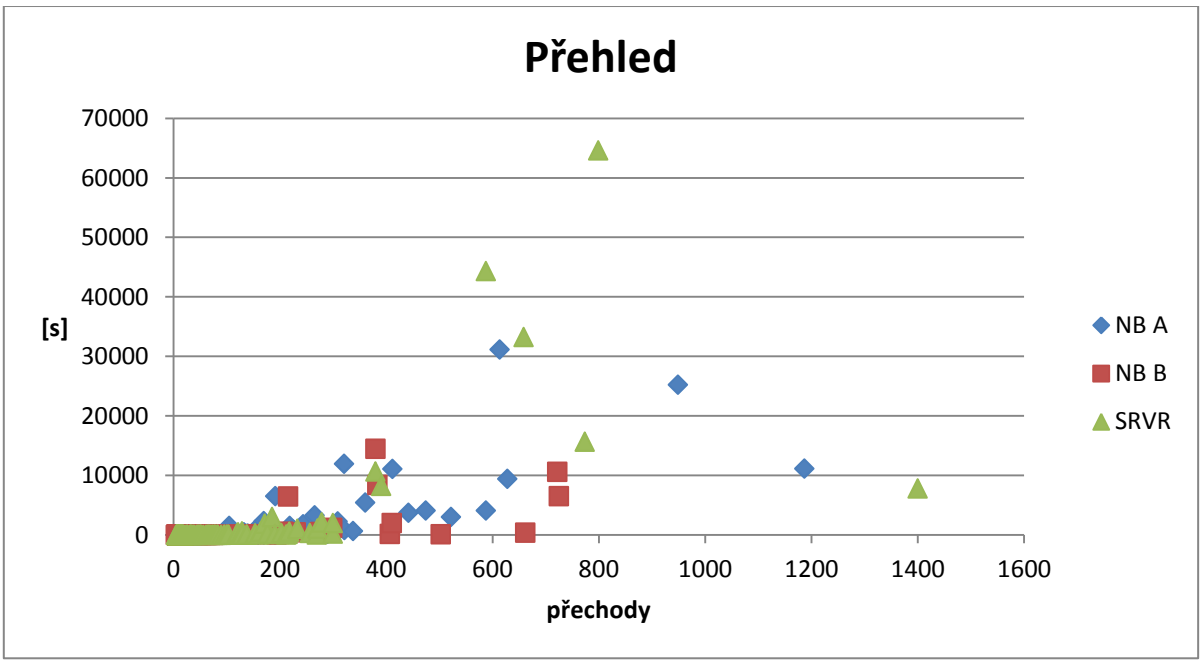
Kdybychom testovali systémy, o kterých víme, že nejsou bisimulačně ekvivalentní, časy by byly výrazně menší i pro velké vstupy. Podmínky ekvivalence, které nejsou tak časově náročné, by byly porušené dřív a tudíž by ani nedošlo k časově tak náročným metodám pro další porovnávání.

Dosažené výsledky vidíme v grafu 5. Do 200 přechodů trval algoritmus na všech zařízeních kolem jedné minuty. Pro více přechodů dosahoval server horší výsledky než notebooky. Notebooky mají srovnatelné výsledky. Horší výsledky na serveru mohly být způsobené prioritou procesu, který měl algoritmus nastaven, jelikož server byl využíván i k dalším operacím.

Z tabulky 1, je patrné, že nejvíce porovnání bylo provedeno na notebooku A. Notebook B průměrně porovnal nejvíce přechodů a míst. Nejhorší čas algoritmu dosahuje průměrně server.

	Počet porovnání	Průměr		
		Místa	Přechody	čas[s]
Notebook A	305	34,89	67,91	482,97
Notebook B	106	839,15	3623,00	759,75
Server	147	40,85	85,80	1355,14

Tabulka 1: přehled zařízení



Graf 5: přehled zařízení

7. Závěr

Myslím si, že všechny úkoly a cíle této diplomové práce se podařilo úspěšně zpracovat a splnit dle požadovaného zadáni.

V první části bylo úkolem nastudovat veškeré teoretické pojmy společně s popisem a průběhem algoritmů na zjišťování bisimilarity. Zde jsem pochopil, co znamená pojem ověřování bisimulační ekvivalence na zadaných vstupních systémech nBPP a nBPA, jak jsou tyto systémy reprezentovány a které veličiny využívají.

Následně bylo nutno se zamyslet nad možností reprezentování daných pojmů v prostředí vybraného programovacího jazyka. Zde jsem volil takový model, který by byl modulární v případě potřeby dalšího rozšiřování programu. Jakmile jsem měl o těchto věcech představu, následoval hlavní cíl celého projektu, a to implementace algoritmů ověřujících bisimilaritu na zadaných vstupech.

Dalším neméně důležitým cílem bylo vytvořit způsoby zadávání těchto vstupních systémů, podle potřeb a parametrů zvolených uživatelem. Jednou z možnosti je generátor náhodných vstupů podle předem určených parametrů, druhou pak načtení vstupů z vhodně definovaného souboru. A samozřejmě možnost zapsat vlastní vstup přímo z klávesnice. Takto lze jednoduše nadefinovat vstupní BPP nebo BPA systém (převážně menšího rozsahu) nebo rozšířit stávající vstupy o další uživatelsky definované přechody.

Posledním splněným cílem je otestování různě velkých vstupů z hlediska časové složitosti algoritmu ověřující bisimulační ekvivalenci na těchto vstupech.

Způsobem, jakým byl nástroj navržen a naimplementován, je možné jej doplnit o algoritmy rozhodující jiné ekvivalence, popřípadě pracující s jinými modely. V podstatě je možné navrhovat nové třídy algoritmů a objektů nebo využívat a rozšiřovat ty stávající. Taktéž je možné generování jiných náhodných vstupů, jelikož je generátor vytvořen nezávisle, jako podsystém, kde takovýchto podsystémů můžeme mít libovolný počet.

Jedinou překážkou při rozšiřování je rozmístění komponent a funkce samotného systému, které by v případě rozšíření o jiné vstupy vyžadovalo dodatečnou modifikaci a přestavbu.

Bc. Martin Bradáč

8. Reference

[1] JANČAR, Petr, Martin KOT a Zdeněk SAWA. Complexity of deciding bisimilarity between normed BPA and normed BPP. *Information and Computation*. 2010, vol. 208, issue 10, s. 1193-1205. DOI: 10.1016/j.ic.2009.10.012.
Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0890540110001069

[2] MARKL Jaroslav, Učební texty k předmětu Petriho sítě I, 26. 9. 2006, VŠB - Technická univerzita Ostrava Fakulta elektrotechniky a informatiky Katedra informatiky. [online] [cit. 2014-03-29]. Dostupné z: http://www.cs.vsb.cz/markl/pn/index.html

[3] DORDA Michal, P/T Petriho sítě, [online] [cit. 2014-02-02], Dostupné z: http://homel.vsb.cz/~dor028/Nekonvencni_metody_2.pdf

[4] JANČAR Petr, KOT Martin, SAWA Zdeněk. Teoretická informatika učební texty, Ostrava 2007, [online] [cit. 2014-02-05]. Dostupné z: http://www.cs.vsb.cz/kot/down/ti2007/TI-text-20070410.pdf

[5] ČERNÝ, Jakub. Silně souvislé komponenty. *Základní grafové algoritmy: algoritmy na grafech jednoduše a srozumitelně* [online]. 2013 [cit. 2014-01-26]. Dostupné z: http://algoritmy.eu/zga/pruchod-grafu/silne-souvisle-komponenty/

[6] MIČKA, Pavel. Tarjanův algoritmus. *Algoritmy.net* [online]. 2010 [cit. 2014-01-26]. Dostupné z: http://www.algoritmy.net/article/1515/Tarjanuv-algoritmus

[7] KOT Martin, JANČAR Petr, SAWA Zdeněk. Normed BPA vs. Normed BPP revisited. November 15, 2008. MEMICS'08, Originally presented at CONCUR'08, Toronto, Canada. [cit. 2014-04-22] [online] Dostupné z: http://www.cs.vsb.cz/kot/download/Publications/2008_Concur_JancarKotSawa_slides.pdf

[8] JANČAR, Petr, Martin KOT. *Bisimilarity on normed Basic Parallel Processes can be decided in time $O(n^3)$*, in: Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems (AVIS 2004), 2004

9. Příloha na CD

Obsah CD:

1) program
    a. determinationbisimilarity – složka obsahující program – zdrojové kódy
    b. documentation – složka obsahující dokumentaci, otevřít lze buď index.html nebo Documentation.chm
    c. manual.pdf – Soubor obsahující uživatelskou příručku k programu
2) text
    a. diplomova_prace_bra284.pdf – soubor obsahující text diplomové práce
    b. timeresults.xlsx – soubor s výsledky časových měření