Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Výpočet stavového prostoru podtřídy barvené Petriho sítě

Jiří Engliš

Diplomová práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří ENGLIŠ**
Osobní číslo:	**I09357**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Výpočet stavového prostoru podtřídy barvené Petriho sítě**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

- V úvodní části práce je nutné provést přehled problematiky barvených Petriho sítí a speciální podtřídy ABA-CPN. - Primárním cílem diplomové práce je návrh a realizace výpočtu kompletního grafu dosažitelných značení (occurrence graph) specifické podtřídy barvené Petriho sítě (ABA-CPN). - Pro prezentační účely bude navržena a implementována vizualizační podpora, která umožní vhodnou zobrazovací formou grafické znázornění stavového prostoru příslušné sítě ABA-CPN.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. JENSEN, K.: Coloured Petri Nets. Basic Concepts, Analysis Method and Practical Use. Volume 1. 1997, ISBN: 3-540-60943-1

2. KAVIČKA, A., KLIMA, V., ADAMKO N.: Agentovo orientovaná simulácia dopravných uzlov. Žilina: EDIS, 2005. 206 s. ISBN: 80-8070-477-5

3. KAVIČKA, A., ŽARNAY, M.: Application of coloured Petri net for agent control and communication in the ABAsim architecture. In Proceeding of 9th workshop and tutorial on practical use of coloured Petri nets and the CPN Tools. K. Jensen (Ed.). Aarhus: University of Aarhus (Denmark), 2008. pp. 47-62. ISSN 0105-8571.

4. CPN Tools home page. [online]. [cited on 29 February 2008] Available at: http://www.daimi.au.dk/ CPNTools/

5. VESELÝ, P.: Interpret barvené Petriho sítě v rámci ABAsim architektury simulačních modelů, diplomová práce, Univerzita Pardubice, 2007.

Vedoucí diplomové práce: **doc. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **27. října 2010**

Termín odevzdání diplomové práce: **20. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 3. listopadu 2010

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 4. 5. 2011

Jiří Engliš

Anotace

Práce se zabývá problematikou architektury simulačních modelů ABAsim a problematikou Pertiho sítí, zvláště podtřídy barvené Petriho sítě zvané ABA-CPN. Dále se zabývá popisem aplikace vyvinuté pro výpočet stavového prostoru této podtřídy barvených Petriho sítí.

Klíčová slova

ABAsim, Petriho síť, ABA-CPN, stavový prostor

Title

Computation of the state space of a subclass of coloured Petri nets

Annotation

The thesis deals with the ABAsim architecture of simulation models and Petri nets, especially a subclass of coloured Petri net called ABA-CPN. It also explains the application that was designed for computing the state space of this coloured Petri net subclass.

Keywords

ABAsim, Petri net, ABA-CPN, state space

Obsah

Seznam ilustrací

1 Úvod a cíle práce

Simulace je velmi využívaná a rozvíjející se disciplína, zvláště v oblastech podpory rozhodování. Umožňuje zkoušet varianty rozhodnutí za nižší náklady a kratší časová období než vyžaduje realizace těchto variant. V poslední době se rozvíjí architektura simulačních modelů zvaná ABAsim. Pro tuto architekturu je charakteristické, že dílčí úkoly rozděluje mezi samostatně pracující agenty, kde každý agent řeší vlastní úkoly a komunikuje pomocí zpráv s agenty ostatními.

Funkce agenta v rámci simulačního modelu lze vyjádřit mnoha způsoby. Jako nejvhodnější řešení se jeví využití deklarativního popisu pomocí např. Petriho sítí. Pro účel popisování funkcí agentů byla navržena zvláštní podtřída barvených Petriho sítí zvaná ABA-CPN.

Pro editaci klasických barvených Petriho sítí existují nástroje, např. CPN Tools (viz [9]). Oproti barveným Petriho sítím však sítě ABA-CPN obsahují specifická omezení a také mírně odlišnou filosofii v případě sledování instancí značek sítí procházejících. Tyto její vlastnosti zatím nejsou postihnuté žádným specializovaným softwarovým nástrojem. Má diplomová práce má za cíl vytvořit jeden z těchto nástrojů, a sice ten, který dokáže vypočítat stavový prostor této sítě, a provést její propojení s nástrojem pro editaci a evoluci, který jako diplomovou práci tvoří Bc. Petr Nejman.

V první části tohoto dokumentu se pokusím popsat problematiku architektury simulačních modelů ABAsim a Petriho sítí. Ve druhé části pak vysvětlím postupy, které využívám v praktické části své práce, tedy aplikaci pro výpočet stavového prostoru.

2 Architektura ABAsim

Tato kapitola čerpá z monografie Agentovo orientovaná simulácia dopravných uzlov [1] autorů Kavička A., Klima V., Adamko N.

ABAsim je architektura simulačních modelů zvaná agentové orientovaná (Agent-Based Architecture). Simulační modely této architektury se skládají z dílčích jednotek, kterým se říká agenti.

2.1 Agent

Agent je zapouzdřený počítačový systém, který je zasazen do nějakého prostředí, v němž pružně a autonomně působí se záměrem plnění daného cíle. Vyznačuje se těmito hlavními vlastnostmi:

- autonomností, neboli schopností pracovat samostatně bez vnějších intervencí,
- společenským chováním, neboli schopností komunikace s ostatními agenty, popřípadě s člověkem,
- reaktivitou, neboli schopností reagovat na vnější podněty,
- iniciativností, neboli schopností vyvíjet cílené chování samostatnou iniciativou, nejen reakcemi na okolí.

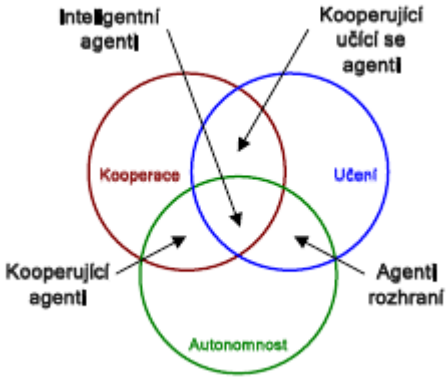
Realizace agentů se podle definice připouští jak hardwarové, tak softwarové. V této práci budu dále rozebírat pouze agenty softwarové. Implementace agentů se mohou lišit také přístupem, kdy výše zmíněným vlastnostem přikládáme různou váhu.

2.1.1 Klasifikace agentů

Agenty lze dělit podle různých parametrů. Jedním parametrem je mobilita agenta. Toto je významné pro simulaci distribuovanou v počítačové síti. Agenty podle tohoto parametru dělíme na statické a mobilní, podle toho, zda zůstávají na jednom uzlu sítě či se mohou přemisťovat.

Dalším parametrem je míra iniciativnosti. Agenty podle tohoto parametru dělíme na reaktivní a uvažující. Uvažující agenti dokážou samostatně rozhodnout o vykonání akce na základě logického uvažování a existenci vnitřního modelu svého okolí, zatímco reaktivní agenti pracují na principu odezvy na podnět a nedisponují interním modelem svého prostředí.

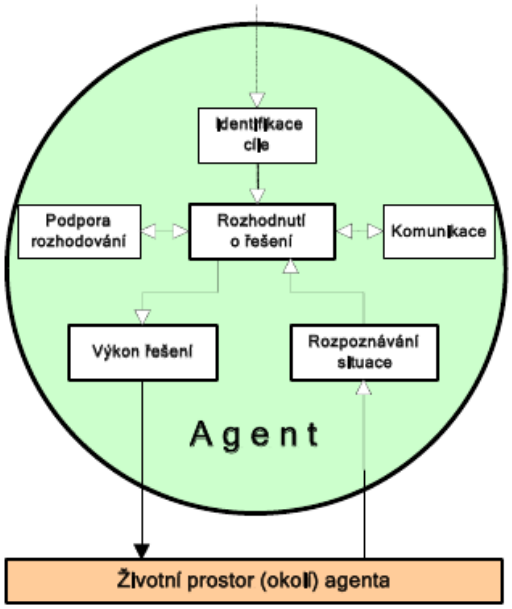
Třetí parametr je založen na hlavních vlastnostech agenta, tedy kooperaci, učení a autonomnost. Podle tohoto kritéria můžeme dostat čtyři typy agentů, a to agenty kooperující, kooperující učící se agenty, agenty rozhraní a inteligentní agenty. Toto dělení popisuje diagram na obrázku 1.



Obrázek 1 – Klasifikace agentů [1]

Tyto parametry patří mezi základní. Další klasifikace jsou prováděny na základě konkrétního aplikačního poslání agenta či podle koncepčního přístupu použitého při jeho konstrukci. Podle jedné ze všeobecně přijímaných klasifikací můžeme agenty rozdělit na následující typy:

- kooperativní agenti,
- agenti rozhraní,
- mobilní agenti,
- informační/internetoví agenti,
- reaktivní agenti,
- hybridní agenti,
- inteligentní agenti.



Obrázek 2 – Funkce agenta [1]

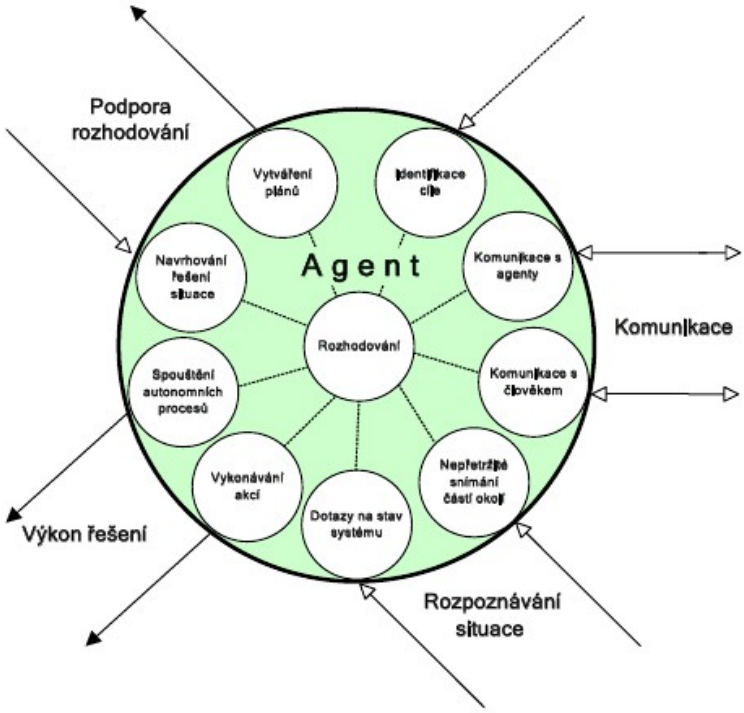
Architektura ABA-SIM byla inspirována agenty reaktivními, proto je dále zaměřím převážně na tento typ.

Reaktivní agenti jsou agenti, kteří nedisponují vnitřním modelem svého okolí, ale pouze reagují na vnější podněty. Jejich funkce připomíná spíše senzorické systémy než systémy s umělou inteligencí. Nevytváří se pro ně plán budoucího chování, jejich činnost je vždy odvozena od aktuální situace, proto se také označují jako agenti situační.

2.1.2 Dekompozice agentů

Každý agent při své funkci využívá podpůrných funkcí. Tyto funkce zprostředkovávají dílčí moduly. Pro každý modul je vhodné, aby agentovi poskytoval schopnosti související s jedinou funkcí. Komunikace mezi těmito moduly je minimální a na nízké úrovni.

Funkce a schopnosti agenta jsou znázorněny na obrázcích 2, respektive 3.



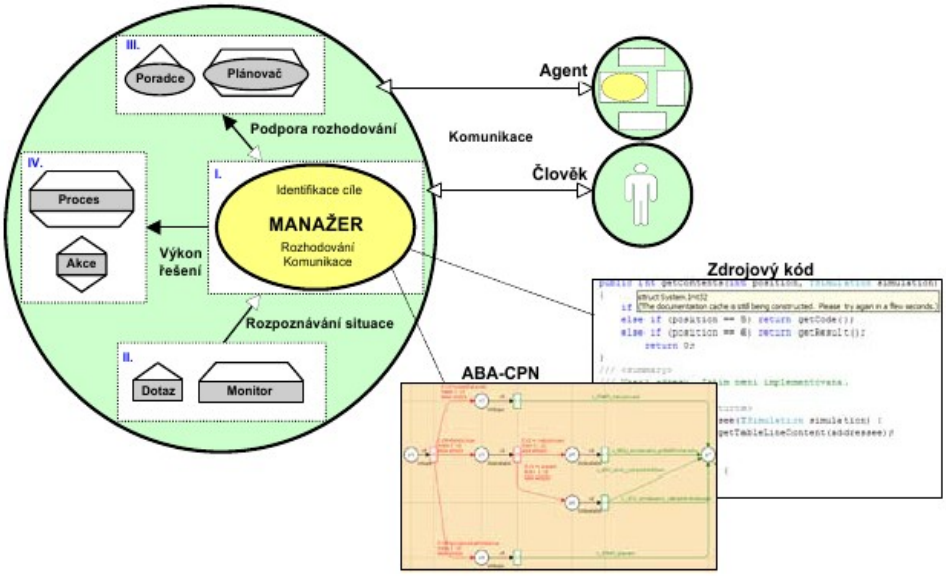
Obrázek 3 – Schopnosti agenta [1]

Agent potřebuje znát cíl, ke kterému se při své funkci přibližovat. Ve svém životním cyklu pak potřebuje umět rozpoznat situaci, zjištěnou buď jednorázově nebo dlouhodobým sledováním okolí. Dále potřebuje rozhodnout o řešení, v čemž mu pomáhají vlastní moduly pro podporu rozhodování. Také může zahájit komunikaci s jinými agenty (případně s člověkem), není-li problém v jeho kompetenci. Pro toto mu slouží modul komunikace. Jakmile má řešení problému, začne jej vykonávat, a to buď jednorázovou akcí nebo dlouhodobým působením na své okolí.

Komponenty v agentech můžeme rozdělit do čtyřech základních skupin:

- senzory, které zpřístupňují informace o stavu systému. Mezi ně řadíme okamžitou komponentu dotaz a dlouhodobou komponentu monitor.
- Řešitelé, kteří podporují rozhodování poskytováním návrhů řešení. Mezi ně řadíme okamžitou komponentu poradce a dlouhodobou komponentu plánovač.
- Efektory, které vykonávají rozhodnutí o změně stavu systému a mění okolí agenta. Mezi ně řadíme okamžitou komponentu akce a dlouhodobou komponentu proces.
- Řídící komponentu, aneb manažer. Tato komponenta zprostředkovává komunikaci ostatních komponent a také je řídí.

Toto dělení vystihuje obrázek 4.



Obrázek 4 – Dekompozice agenta [1]

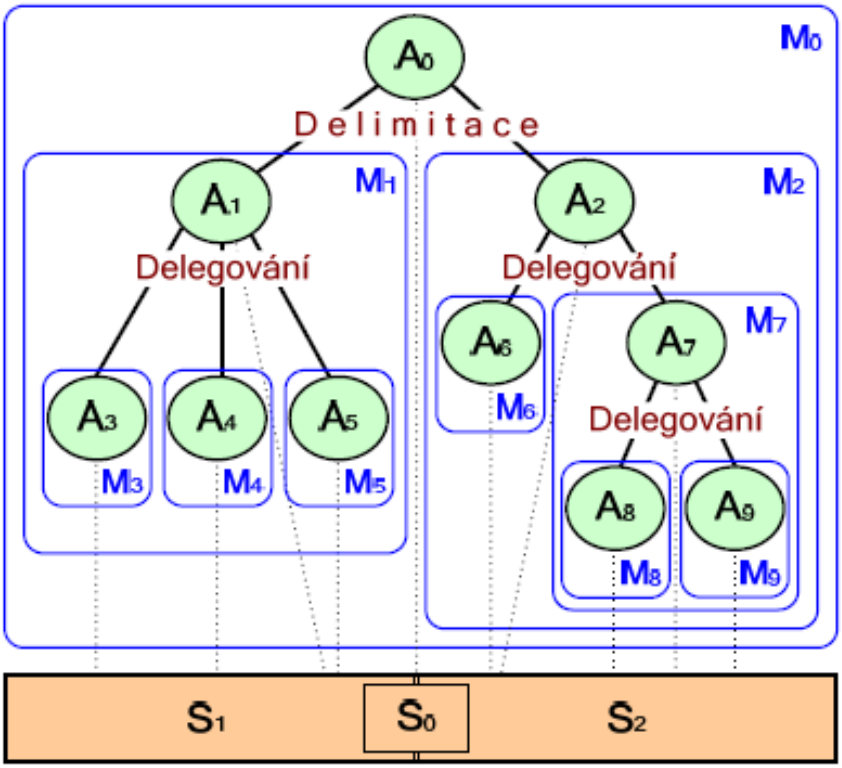
Pro popis vnitřní logiky komponent, zvláště manažerů, lze využít různých způsobů. První možnost je imperativní, kdy se logika implementuje přímo pomocí programovacího jazyka. Výhodnější může být možnost druhá, deklarativní, kdy pro popis využíváme například formalismus Petriho sítí, zvláště podtřídy ABA-CPN, pro toto využití zvlášť vyvíjenou. Problematice Petriho sítí je věnována kapitola 3 .

2.2 Popis architektury

2.2.1 Multiagentový přístup

Pro realizaci jednoduchých simulačních modelů může stačit jediný agent. Často však potřebujeme, aby model vystihoval organizaci řízení jednotlivých částí, obvykle hierarchickou. Takovýto simulační model se pak skládá z více agentů, kde se každý agent stará o jednu z částí.

V multiagentových hierarchických systémech hovoříme o delimitaci a delegování. Tyto pojmy ilustruje obrázek 5.



Obrázek 5 – Delimitace a delegování [1]

Delimitace znamená, že agent rozděluje správu na samostatné dílčí jednotky, přičemž si od nich nechává posílat informace o závažných skutečnostech, případně těmto jednotkám sám informace posílá. Obě dílčí jednotky jsou si rovnocenné a předpokládá se, že provádí stejné správní úkony.

Delegování znamená, že agent k plnění svých cílů využívá agentů specializovanějších, na které deleguje část svých kompetencí.

Multiagentový model můžeme vyjádřit algebraicky. Na strukturu podřízených agentů, tedy množinu A, pohlédneme jako na strukturu modelů, tedy množinu M. Pro tuto množinu platí, že každý model $M_i \in M$, ($M_i \subseteq A$) se skládá ze stromu agentů, jehož kořenem je agent $A_i \in A$, $i = 0$,..., $|A|$-1. Agenta A_i pak nazýváme představitelem modelu M_i a modely na nejnižší úrovni hierarchie jsou modely jednoagentové. Pokud dále zavedeme

hranaté závorky pro označení delimitace a kulaté závorky pro delegování, dostaneme zápis modelu, který v případě obrázku 5 může vypadat následovně: $M_0[M_1(M_3,M_4,M_5),M_2(M_6,M_7(M_8,M_9))]$

2.2.2 Komunikační mechanismus

V architektuře je komunikace prováděna pouze pomocí zpráv, a to jak na úrovni meziagentové, tak na úrovni manažer-asistent.

Zprávy, které slouží pro komunikaci mezi agenty, konkrétně mezi jejich manažery, jsou následující:

- Notice, neboli oznámení. Na tento typ zprávy není očekávána odpověď.
- Request, neboli požadavek. Na tento typ zprávy agent očekává odpověď.
- Response, neboli odpověď. Tento typ zprávy je povinná odpověď na předchozí požadavek.

Zprávy, které slouží pro komunikaci mezi manažerem a asistenty jsou následující:

- Start, neboli začátek. Tato zpráva spouští činnost kontinuálního asistenta.
- Execute, neboli provedení. Tato zpráva provede činnost promptního asistenta.
- Break, neboli přerušení činnosti. Tato zpráva okamžitě ukončí činnost kontinuálního asistenta. V obvyklých případech se tato zpráva nevysílá. Její smysl spočívá hlavně při mimořádném ukončení simulace.
- Notice, neboli oznámení. Tuto zprávu vysílají kontinuální asistenti manažerovi v případě, že jej potřebují upozornit na důležité informace.
- Finish, neboli ukončení činnosti. Tuto zprávu vysílají kontinuální asistenti manažerovi v případě, že dokončili svou činnost.

Kontinuální asistent sám sobě při své činnosti posílá zprávu Hold. V rámci architektury ABAsim je tento typ zprávy jediný, který obsahuje časové razítko a posunuje simulační čas. Kontinuální asistent při přijetí zprávy vykoná činnost a pošle sám sobě další zprávu hold. Než tuto zprávu přijme, je neaktivní.

3 Petriho sítě

Následující pasáž a podkapitola 3.1 převážně čerpají z monografie Petriho sítě [2] autora Češka, M.

Petriho síť je označení pro širokou třídu diskrétních matematických modelů, které umožňují specifickými prostředky popsat řídící a informační toky uvnitř modelovaných systémů.

Trojici $N = (P, T, F)$ lze nazývat sítí, pokud $P \cap T = \emptyset$ a $F \subseteq (P \times T) \cup (T \times P)$, kde

- P je konečná množina míst sítě N,
- T je konečná množina přechodů sítě N,
- F je toková relace sítě N.

Petriho síť lze znázornit pomocí orientovaného grafu, který vznikne grafovou reprezentací relace F. Z definice můžeme vidět, že vrcholy grafu sítě tvoří dva rozdílné typy, a to místa a přechody. Místa se graficky značí kruhy, přechody pak obdélníky. Dále můžeme vidět, že prvky tokové relace, reprezentované orientovanými hranami, mohou vést pouze od místa k přechodu nebo od přechodu k místu.

Petriho sítí existuje mnoho tříd. V souvislosti s tímto mluvíme o rozšířeních Petriho sítí, pokud základní třídu rozšiřují, či o podtřídách Petriho sítí, pokud základní třídu zužují. Rozšíření zvyšují modelovací mocnost, zatímco podtřídy obvykle poskytují větší rozhodovací mocnost. V této práci postupně proberu nejzákladnější třídy sítí a dále se budu zabývat převážně podtřídou barvených Petriho sítí ABA-CPN.

3.1 Klasická Petriho síť

3.1.1 Definice

Také známá pod označením P/T Petriho síť. Jedná se o nejjednodušší třídu Petriho sítí.

Šestici $PN = (P, T, F, W, K, M_0)$ nazýváme P/T Petriho sítí, jestliže

- (P, T, F) je konečná síť,
- W je ohodnocení hran grafu sítě určující váhu každé hrany,
- K je zobrazení specifikující kapacitu každého místa,
- M_0 je počáteční značení míst sítě.

Pro W platí $W : F \rightarrow N\text{-}\{0\}$. Určuje, kolik značek každá hrana přenáší při provedení incidentního přechodu.

Pro K platí $K : P \rightarrow N \cup \{\infty\}$. Určuje, kolik značek se může najednou maximálně vyskytovat na místě.

Pro M_0 platí $M_0 : P \rightarrow N \cup \{\infty\}$ a $\forall p \in P : M_0(p) \leq K(p)$. Určuje, kolik značek se vyskytuje na místech při inicializaci sítě. Respektuje kapacity míst.

3.1.2 Chování sítě

Chování sítě popisují evoluční pravidla. Pro popis chování zavádíme pojem značení Petriho sítě. To je zobrazení $M : P \rightarrow N \cup \{\infty\}$ pro které platí $p \in P : M(p) \leq K(p)$. Dále zavádíme označení $'t$ a t' pro vstupní, respektive výstupní okolí přechodu. Platí $\forall t \in T$:

- $'t = \{p \mid pFt\}$,
- $t' = \{p \mid tFp\}$.

Tvrdíme, že přechod t je proveditelný při značení M (M-proveditelný), pokud každé místo v jeho vstupním okolí obsahuje tolik značek, kolik je váha hrany s přechodem t a místem incidentní a pokud každé místo v jeho výstupním okolí má takovou kapacitu, aby se do něj dalo vložit tolik značek, kolik je váha hrany s přechodem t a místem incidentní, neboli:

- $\forall p \in 't : M(p) \geq W(p,t)$,
- $\forall p \in t' : M(p) \leq K(p) - W(t,p)$,

Pokud je přechod t M-proveditelný, získáváme ke značení M následné značení M', které získáme podle následujících pravidel. Z každého místa ve vstupním okolí přechodu t se odebere tolik značek, kolik jsou váhy hran s místem a přechodem t incidentní a do každého místa ve vstupním přechodu t se vloží tolik značek, kolik jsou váhy hran s místem a přechodem t incidentní. Tedy $\forall p \in P$:

- pokud $p \in 't \setminus t'$, pak $M'(p) = M(p) - W(p,t)$.
- Pokud $p \in t' \setminus 't$, pak $M'(p) = M(p) + W(t,p)$.
- Pokud $p \in 't \cap t'$, pak $M'(p) = M(p) - W(p,t) + W(t,p)$.
- Jinak $M'(p) = M(p)$.

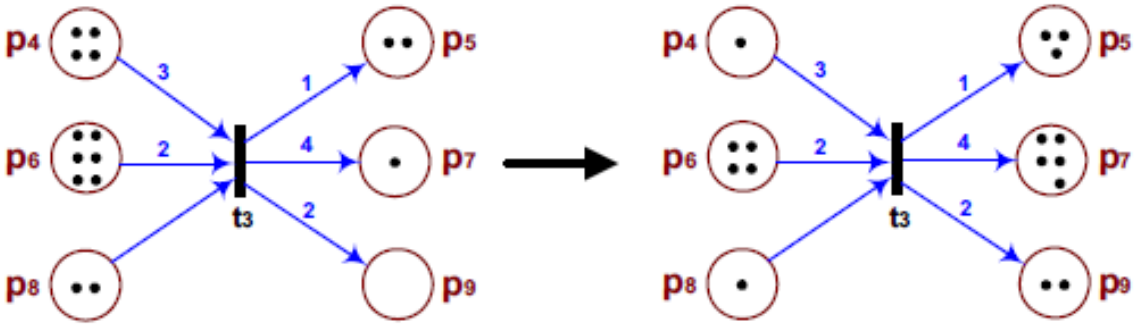
Symbolicky lze provedení přechodu zapsat jako $M\ [\ t > M'$.

Dále zavádíme množinu $[\ M >$ jako množinu dosažitelných značení ze značení M. Pak platí $M \in [\ M >$ a je-li $M_1 \in [\ M >$ a pro nějaké $t \in T$ platí $M_1\ [\ t > M_2$, pak $M_2 \in [\ M >$. Množina $[\ M_0 >$ pak je množina dosažitelných značení sítě PN.

3.1.3 Příklady chování sítě

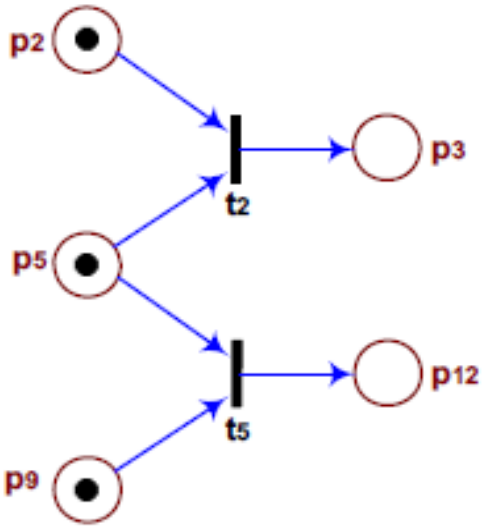
Na obrázku 6 je příklad, jak získáme značení M' ze značení M při provedení přechodu t_3. Je patrné, že váha W nepopsaných hran se považuje za váhu jednotkovou a kapacita K nepopsaných míst je ∞. Dále je patrné, že při provádění přechodu jsou ovlivněna pouze ta místa, které se nacházejí bezprostředně před nebo bezprostředně za přechodem, tedy ty, které spadají do množin $'t$ a t'.

V síti mohou nastat takové případy, kdy je proveditelný více než jeden přechod. Při evoluci sítě se v takovém případě náhodně vybere jeden z nich. Každý z proveditelných přechodů má stejnou šanci na to, že bude vybrán. Pokud jsou přechody t_1 a t_2 proveditelné a platí $('t_1 \cup t_1') \cap ('t_2 \cup t_2') = \varnothing$, říkáme, že přechody t_1 a t_2 jsou prováděny nezávisle na sobě, neboli paralelně.



Obrázek 6 – Příklad provedení přechodu [7]

Obrázek 7 ukazuje situaci, kdy jsou dva přechody proveditelné, nikoliv však paralelně, a zároveň ukazuje možnost, jak lze Petriho sítě použít pro vyjádření soupeření o prostředky.



Obrázek 7 – Příklad soupeření o prostředky

Při provedení přechodu t_2 se odebere značka z míst p_2 a p_5, takže přechod t_5 již proveditelný nebude. Obdobně při provedení přechodu t_5 již nebude proveditelný přechod t_2. Výběr přechodu, který bude proveden, je náhodný. Oba přechody mají stejnou pravděpodobnost, že budou provedeny.

3.2 Barvené Petriho sítě

Tato podkapitola čerpá z monografie Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use [3] autora Jensen, K.

Barvené Petriho sítě jsou rozšířením P/T Petriho sítí. U značek, které sítí prochází, navíc sledujeme jejich typ, neboli barvu. Tohoto rozšíření lze velmi dobře využít např. pro možnosti rozhodovacího větvení.

3.2.1 Definice

Barvenou Petriho sítí nazýváme devítici $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, ve které:

- Σ je konečná množina neprázdných typů, také nazývaných množina barev,
- P je konečná množina míst,
- T je konečná množina přechodů,
- A je konečná množina hran,
- N je funkce vrcholů,
- C je funkce barev,
- G je funkce strážních podmínek,
- E je funkce hranových výrazů,
- I je inicializační funkce.

K výše uvedeným množinám a funkcím dále uvádím podrobnější popis:

Množina Σ určuje hodnoty údajů, operací a funkcí, které je možné použít v síťových výrazech. Předpokládáme, že každý typ obsahuje alespoň jeden prvek. Prvky mohou být jednoduché, nebo strukturované.

Pro P, T a A platí $P \cap T = P \cap A = T \cap A = \emptyset$. Jelikož jsou množiny konečné, nemůže nastat situace, kdy mezi dvěma vrcholy existuje nekonečný počet hran.

Pro N platí $N : A \rightarrow (P \times T) \cup (T \times P)$. Funkce mapuje každou hranu na uspořádanou dvojici vrcholů hrany, kde první je zdrojový a druhý cílový. Cílový vrchol musí být jiného druhu než zdrojový. Podle definice existuje možnost definovat mezi dvojicí vrcholů více hran.

Pro C platí $C : P \rightarrow \Sigma$. Funkce zobrazuje každé místo p na typ $C(p)$. Každá značka v místě p tedy musí mít hodnotu typu $C(p)$.

Pro G platí $G : T \rightarrow \{\forall t \in T : (Type(G(t)) = B \wedge Type(Var(G(t))) \subseteq \Sigma)\}$. Funkce převádí každý přechod t do booleovského výrazu, ve kterém všechny proměnné jsou typu z množiny Σ. Je-li výsledek výrazu vždy *true*, výraz se neuvádí.

Pro E platí $E : A \rightarrow \{\forall a \in A : (Type(E(a)) = C(p) \wedge Type(Var(E(a))) \subseteq \Sigma)\}$. Funkce mapuje každou hranu na výraz typu $C(p)_{MS}$. To znamená, že každý hranový výraz se musí vyhodnotit do multimnožin nad typem přilehlého místa p. Diagram *CPN* může obsahovat i hranový výraz *expr* typu $C(p)$, což se považuje za zkrácení zápisu 1`(*expr*).

Pro I platí $I : P \rightarrow \{\forall p \in P : Type(I(p)) = C(p)_{MS}\}$. Funkce zobrazuje každé místo p na uzavřený výraz, který musí být typu $C(p)_{MS}$. Je-li inicializační výraz vyhodnocen jako ∅, tak se vynechává.

3.2.2 Chování sítě

Pro popis chování sítě si zadefinujeme následující notaci, kde $t \in T$ je přechod a $(x_1,x_2) \in (P \times T) \cup (T \times P)$ je dvojice vrcholů:

- $A(t) = \{\forall a \in A \mid N(a) \in (P \times \{t\}) \cup (\{t\} \times P)\}$ pro množinu všech hran incidentních s přechodem t,
- $Var(t) = \{v \mid v \in Var(G(t)) \vee \exists\, a \in A(t) : v \in Var(E(a))\}$ pro množinu všech proměnných, které se nacházejí ve strážní podmínce přechodu t nebo na jeho incidentní hraně,
- $A(x_1, x_2) = \{a \in A \mid N(a) = (x_1, x_2)\}$ pro množinu hran mezi vrcholy x_1 a x_2.
- $E(x_{1,} x_2) = \sum_{a \in A(x_{1,} x_2)} E(a)$ pro součet multi-množin z výrazů na hranách mezi vrcholy x_1 a x_2.

Dále definujeme:

- $B(t)$ jako množinu všech vazeb pro t, kde vazba přechodu t je funkce b definovaná na $Var(b)$, po níž platí

  $\forall v \in Var(t) : b(v) \in Type(v)$,

  $G(t) < b >$,

- BE jako množinu všech vazbových prvků, kde vazbový prvek je dvojice (t,b), pro níž platí $t \in T$ a $b \in B(t)$,
- TE jako množinu značkových prvků, kde značkový prvek je dvojice (p,c), pro níž platí $p \in P$ a $c \in C(p)$,
- M, respektive Y jako množinu značení, respektive kroků, kde značení je multimnožina nad TE, pokud krok je neprázdná a konečná multimnožina nad BE.

Počáteční značení pak je značení, které získáme vyhodnocením inicializačních výrazů

$$\forall (p, c)\ TE : M_0\,(p, c) = (I(p))(c).$$

Značení se často reprezentuje funkcemi definovanými na množině P. Každé značení $M \in TE_{MS}$ určuje jedinečnou funkci M^* definovanou na množině P takovou, že $M^*(p) \in C(p)_{MS}$:

$$\forall p \in P\ \forall c \in C(p) : (M^*(p))(c) = M(p, c).$$

Každá funkce M^* definovaná na množině P taková, že $M^*(p) \in C(p)_{MS}$ pro všechny $p \in P$, určuje jedinečné značení M:

$$\forall (p, c) \in TE : M(p, c) = (M^*(p))(c).$$

Je-li splněna vlastnost $\forall p \in P: \sum_{(t,b)\in Y} E(p,t)\langle b\rangle \leq M(p)$, říkáme, že je aktivován krok Y při značení M. Pokud $(t,b) \in Y$, říkáme, že přechod t je aktivovaný při značení M pro vazbu b nebo že je aktivovaný vazbový prvek (t,b) při značení M. Vazbové prvky (t_1,b_1) a (t_2,b_2) jsou aktivované souběžně, pokud $(t_1,b_1),(t_2,b_2) \in Y \wedge (t_1,b_1) \neq$ Přechod t, respektive vazbový prvek(t,b) je souběžně aktivovaný sám sebou, pokud $|Y(t)| \geq 2$, respektive $|Y(t,b)| \geq 2$.

Je-li krok Y aktivovaný při značení M_1, může se vykonat. Při tom změní značení M_1 na značení M_2, pro nějž platí:

$$\forall p \in P: M_2(p) = \left(M_1(p) - \sum_{(t,b)\in Y} E(p,t)\langle b\rangle\right) + \sum_{(t,b)\in Y} E(p,t)\langle b\rangle$$

Odečítané sumě se říká odstraněné značky, přičítané sumě se říká přidané značky.

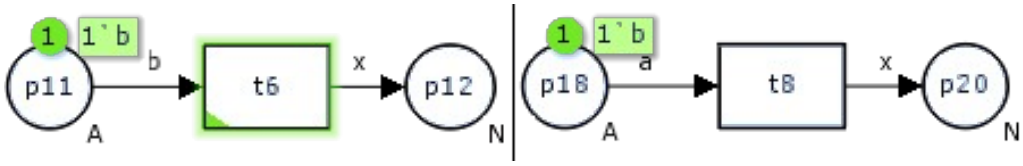
Říkáme, že značení M_2 je přímo dosažitelné ze značení M_1 výskytem kroku Y a zapisujeme $M_1 [Y > M_2$.

Definujeme posloupnost vykonávání přechodů, což je posloupnost značení a kroků, která může být nekonečná nebo konečná. Pro konečnou posloupnost platí $M_i [Y_i > M_{i+1}$, kde $i \in \{1, 2, \ldots, n\}$ a $n \in N$. M_1 je počáteční značení, M_{n+1} je koncové značení a n je délka. Pro nekonečnou posloupnost platí $M_i [Y_i > M_{i+1}$, kde $i \geq 1$, $i \in N$. Existuje-li konečná posloupnost značení, která začíná v M' a končí v M'', říkáme, že značení M'' je dosažitelné ze značení M'. Množinu značení dosažitelných z M' označujeme $[M' >$. Pokud značení spadá do množiny $[M_0 >$, je dosažitelné.

3.2.3 Příklady chování sítě

Pro následující příklady si zadefinujme množiny barev $A = \{a,b\}$ a $N = \{x,y\}$. Dále si zadefinujme proměnné *xa*, *xb* typu *A* a *xn* typu *N*.

Hranové výrazy barvených Petriho sítí se mohou skládat z libovolného počtu proměnných a konstant, případně rozhodovacích výrazů ve tvaru *if proměnná=barva then výraz else výraz*, kde za *výraz* lze dosadit libovolný počet proměnných a konstant.



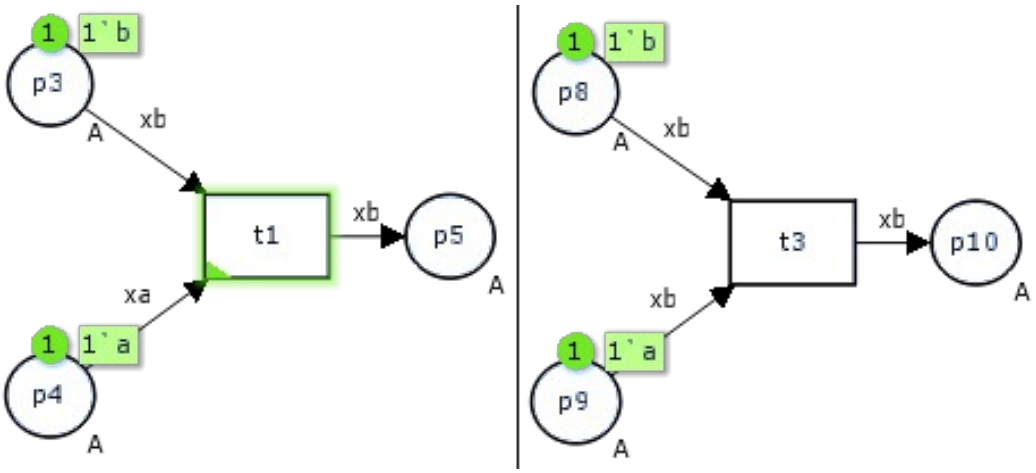
Obrázek 8 – Příklad konstant CPN

Obrázek 8 ukazuje příklady částí sítě, kde přechod s místy spojuje hrana s konstantou v hranovém výrazu. V levé části obrázku hrana mezi místem p_{11} a přechodem t_6 propouští pouze značky s barvou b. Protože v místě p_{11} je značka s barvou b, je přechod t_6 proveditelný.

Při proveden přechodu se značka z místa p_{11} odebere a na místo p_{12} se vloží značka s barvou x, protože hrana mezi přechodem t_6 a místem p_{12} přenáší konstantu x.

V pravé části obrázku obsahuje místo p_{18} značku s barvou b, ale mezi ním a přechodem t_8 je hrana, která propouští pouze značky s barvou a. Proto není přechod t_8 proveditelný.

Obrázek 9 ukazuje příklady částí sítě, kde přechod s místy spojuje hrana s proměnnou v hranovém výrazu. Proměnné *xa* a *xb* přenáší libovolnou značku s barvou, která spadá do množiny barev A, v tomto případě barvy a a b. Protože ze všech míst z množiny $'t_1$ hrany značky propustí, je přechod t_1 proveditelný.



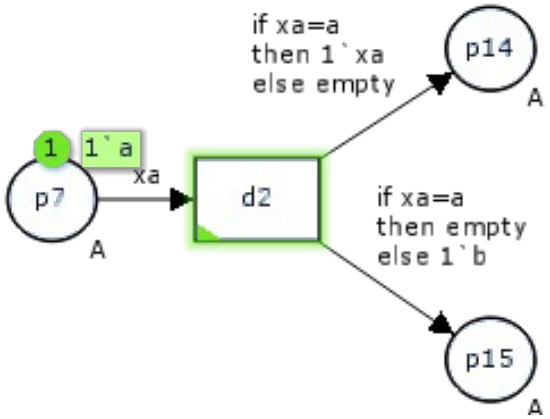
Obrázek 9 – Příklad proměnných CPN

Při provedení přechodu jsou odebrány značky z míst z množiny $'t1$. Vyskytuje-li se více značek, které mají vyhovující barvu, než kolik jich specifikuje hranový výraz, je náhodně vybráno tolik značek, kolik jich hranový výraz specifikuje. Všechny značky mají stejnou pravděpodobnost, že budou odebrány. Na místa z množiny t_1' (v tomto případě p_5) se vloží značky, jejichž barva odpovídá značkám barvy odebrané z přechodů z množiny $'t_1$, které byly odebrané stejnou proměnnou. Pokud proměnná mezi přechodem a místem z t' není stejná, vloží se značky s libovolnou barvou z odpovídající množiny barev. Všechny barvy mají stejnou pravděpodobnost, že budou vybrány.

Pravá část obrázku ukazuje, že pokud mezi více místy z $'t$ a přechodem t existuje více hran se stejnou proměnnou, je přechod proveditelný pouze pokud mají značky v daných místech stejnou barvu.

Obrázek 10 ukazuje použití rozhodovacích přechodů a hran. Rozhodovací hrana vyhodnocuje, zda má značka z místa z $'t$, přenesená proměnnou uvedenou v podmínce, takovou barvu, jaká je uvedena v podmínce. Podle tohoto logického výrazu pak vybere buď výraz za klíčovým slovem *then*, pokud je podmínka vyhodnocena jako *true*, nebo výraz za klíčovým slovem *else*, je-li podmínka vyhodnocena jako *false*.

Na příkladu přenáší proměnná *xa* značku s barvou *a*, tudíž pro obě rozhodovací hrany je výraz vyhodnocen na *true*. Značka umístěná na místo p_{14} pak má barvu *a*, kvůli shodě proměnné hran před a za přechodem. Na místo p_{15} se neumístí žádná značka kvůli klíčovému slovu *empty*. Obdobně by se na místo p_{14} neumístila žádná značka a na místo p_{15} by se umístila značka s barvou *b*, kdyby značka přenášená proměnnou *xa* neměla barvu *a*.



Obrázek 10 – Příklad rozhodovacího přechodu

3.3 Podtřída ABA-CPN

Tato podkapitola čerpá převážně z článků [4] a [8] autorů Kavička, A., Žarnay, M.

Jak bylo zmíněno v kapitole 2.1.2 , Petriho sítě poskytují vhodný formalismus pro popis chování manažera agentů v architektuře ABAsim. Pro tento účel byla zvlášť vyvinuta podtřída barvených Petriho sítí zvaná ABA-CPN. Chování této podtřídy je obdobné chování klasickým barveným Petriho sítím, jsou však kladeny dodatečné podmínky na její konstrukci a má vlastní konvenci značení.

3.3.1 Specifikace

Síť ABA-CPN je síť $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, kde:

- Σ je konečná množina neprázdných typů, také nazývaných množina barev,
- P je konečná množina míst,
- T je konečná množina přechodů,
- A je konečná množina hran,
- N je funkce vrcholů,
- C je funkce barev,
- G je funkce strážních podmínek,
- E je funkce hranových výrazů,
- I je inicializační funkce.

Definice těchto množin a funkcí vychází z definice třídy barvených Petriho sítí uvedené v kapitole 3.2.1 . V následujícím textu jsou popsány ty množiny a funkce, jejichž definice je pozměněna nebo rozšířena.

Množina P je pro lepší rozlišitelnost specifických druhů míst rozdělena na podmnožiny $P = \{p_{in}\} \cup \{p_{out}\} \cup P_S$. Tyto podmnožiny jsou vzájemně disjunktní. Místo p_{in} nazýváme vstupním místem, místo p_{out} nazýváme výstupním místem. Ostatní místa, která spadají do množiny P_S, nazýváme místy interními. Je-li značka na vstupním místě, znamená to, že agent přijal zprávu. Je-li značka na výstupním místě, znamená to, že je agent připraven odeslat zprávu jinému agentovi.

Prvky z množiny P mají další vlastnosti:

- $(\forall t \in T: \neg\exists a \in A: N(a) = (t, p_{in})) \wedge (\exists_1 a \in A: N(a) = (p_{in}, t), t \in T)$,
- $(\exists a \in A: N(a) = (t, p_{out}), t \in T) \wedge (\forall t \in T: \neg\exists a \in A: N(a) = (p_{out}, t))$,
- $\forall p \in P_S: (\exists a \in A: N(a) = (t, p), t \in T) \wedge (\exists_1 a \in A: N(a) = (p, t), t \in T)$.

Tyto vlastnosti specifikují, že do vstupního místa není zaústěná žádná hrana a je incidentní právě s jednou výstupní hranou, z výstupního místa nevede žádná hrana a je incidentní alespoň s jednou vstupní hranou a ostatní místa mají vždy alespoň jednu vstupní hranu a právě jednu hranu výstupní.

Množina T je rozdělena na podmnožiny $T = T_D \cup T_A \cup T_S \cup T_B$, $T \neq \emptyset$. Prvky podmnožiny T_D nazýváme rozhodovacími přechody a představují body podmínkového větvení. Prvky podmnožiny T_A nazýváme asistenčními přechody a představují interní výkonné komponenty agentů. Prvky podmnožiny T_S nazýváme odesílacími přechody a představují model pro zasílání zpráv komponentům nebo jiným agentům. Prvky podmnožiny T_B nazýváme standardními přechody a vzhledem k simulační architektuře nemají specifickou interpretaci.

Prvky z množiny T mají další vlastnosti:

- $\forall t \in T_B: \exists a \in A: N(a) = (p, t), p \in P$,
- $\forall t \in T \setminus T_B: \exists_1 a \in A: N(a) = (p, t), p \in P$,
- $\forall t \in T_D \cup T_S : \exists a \in A: N(a) = (t, p), p \in P$,
- $\forall t \in T: G(t) = \emptyset$,
- vstupním přechodem nazýváme přechod t, pro který platí:

  $\exists_1 t \in T_D: \exists_1 a \in A: N(a) = (p_{in}, t)$,
- výstupním přechodem nazýváme přechod t, pro který platí:

  $t \in T: (\exists a \in A: N(a) = (t, p_{out})) \vee (\neg\exists a \in A: N(a) = (t, p), p \in P)$,

- interním přechodem nazýváme přechod t, pro který platí:

  $t \in T$: $(\neg\exists a \in A: N(a) = (p_{in}, t)) \wedge (\neg\exists a \in A: N(a) = (t, p_{out})) \wedge (\exists a \in A: N(a) = (t, p), p \in P)$.

Tyto vlastnosti specifikují, že přechody z množiny T_B, tedy standardní přechody, jsou incidentní s alespoň jednou vstupní hranou a ostatní přechody jsou incidentní s právě jednou vstupní hranou. Zároveň jsou přechody z množin T_S a T_D, tedy odesílací a rozhodovací přechody, jsou incidentní s alespoň jednou výstupní hranou. Žádný z přechodů nedisponuje strážní podmínkou.

Hrany z množiny A lze rozdělit na několik kategorií. Pokud $t(a) \in T_D \wedge N(a) \in T \times P$, tedy pokud hrana vychází z rozhodovacího přechodu, hranu a nazýváme rozhodovací hranou a hranový výraz $E(a)$ obsahuje podmínku pro variantní větvení. Pokud $t(a) \notin T_D$ a všechna $t(a) \in T_D : N(a) \in P \times T$, pak pokud je hranový výraz $E(a)$ složen pouze z jedné konstanty, nazýváme hranu a konstantní hranou a pokud je hranový výraz $E(a)$ složen pouze z jedné proměnné, nazýváme hranu a elementární hranou.

Neexistuje dvojice hran $a_1, a_2 \in A$ $a_1 \neq a_2$, pro kterou by platilo $p(a_1) = p(a_2) \wedge t(a_1) = t(a_2) \wedge ((N(a_1) \in (T \times P) \wedge N(a_2) \in (P \times T))$, kde $p(a)$ znamená místo a $t(a)$ znamená přechod z $N(a)$. Tato specifikace znamená, že v sítích ABA-CPN není povolena konstrukce vlastních cyklů. Síť ABA-CPN tedy vždy představuje čistou síť.

Petriho síť vybudovaná z prvků množin P, T a A reprezentuje acyklickou síťovou strukturu.

Množina přípustných inicializačních značení $M_0 \subset I(P)$ je definována následovně: $M_0 = \{{}^jM_0 \mid j = 1,2, \ldots, |C(p_{in})|\}$, kde ${}^jM_0$ označuje j-té inicializační značení, pro které platí: $\forall p \in P$: $|{}^jM_0(p)| = 1$, jestliže $p = p_{in}$ a $|{}^jM_0(p)| = 0$, jestliže $p \neq p_{in}$; pro $i \neq j$ platí: ${}^iM_0(p) \neq {}^jM_0(p)$. V rámci ABA-CPN odrážejí značky různých barev odlišně vyplněné formuláře zpráv, které se v architektuře ABAsim využívají pro komunikační účely. Přípustné inicializační značení sítě představují situaci, kdy vstupní zpráva čeká na své zpracování a agent zrovna nezpracovává jinou zprávu, tedy v síti se nevyskytují značky, s výjimkou vstupního místa, které obsahuje právě jednu značku, která má barvu z množiny barev vstupního místa $C(p_{in})$. Tato množina obsahuje barvy, které představují zprávy, které je agent schopný zpracovat, proto $|M_0| = |C(p_{in})|$.

ABA-CPN také klade požadavky na mrtvé značení. To je značení, při kterém již není proveditelný žádný přechod, tedy při něm došlo k ukončení evoluce sítě. Přípustné mrtvé značení je takové značení, pro které platí $|M(p_j)| = 0 \wedge |M(p_n)| \geq 0$ pro $j = 1, \ldots, n\text{-}1$ kde $n = |P|$. To znamená, že po skončení evoluce je výskyt značek přípustný pouze ve výstupním místě.

3.3.2 Konvence pro popis sítí

Pro sítě ABA-CPN byly navrženy konvence pro pojmenovávání míst, přechodů a konstrukci hranových výrazů, aby bylo možné provedení jednoznačné transformace sítě do datových struktur a její korektní evoluce v rámci interpretů a simulačních programů.

- Místa označujeme p_i, kde $i = 1, \ldots, n$ a $n = |P|$. Místo p_1 představuje vstupní místo a p_n výstupní místo.
- Přechody označujeme podle podmnožiny, do které spadají. Rozhodovací přechody označujeme d_i, kde $i = 1, \ldots, n$ a $n = |T_D|$. Asistenční přechody označujeme a_i, kde $i = 1, \ldots, n$ a $n = |T_A|$. Odesílací přechody označujeme s_i, kde $i = 1, \ldots, n$ a $n = |T_S|$. Standardní přechody označujeme t_i, kde $i = 1, \ldots, n$ a $n = |T_B|$.
- Hranový výraz elementárních hran sestává pouze z jednoho symbolu, který představuje proměnnou.
- Hranový výraz konstantních hran sestává pouze z jednoho symbolu, který představuje proměnnou nebo konkrétní barvu.
- Hranový výraz rozhodovacích hran sestává ze tří částí. První část tvoří klíčové slovo if následované podmínkou ve tvaru „*proměnná=hodnota*“. Druhou část tvoří klíčové slovo *then* následované výrazem, který je tvořen pouze jedním symbolem a obsahuje barvu, konstantu nebo proměnnou, popřípadě klíčové slovo *empty*. Třetí část tvoří klíčové slovo *else* následované výrazem, který je tvořen pouze jedním symbolem a obsahuje barvu, konstantu nebo proměnnou, popřípadě klíčové slovo *empty*.

Dále byly navrženy konvence pro identifikátory proměnných a konstant. Pomocí nich může návrhář sítě jednak ovlivňovat evoluci sítě vzhledem k „datovému obsahu“ značky, tak ovlivňovat instance jednotlivých značek a jejich průchod danou sítí. Pojem instance značky, popřípadě formuláře zprávy, je pro třídu ABA- CPN specifický. V klasických sítích značky po odebrání z míst zanikají a při vkládání na místa se tvoří nové. Toto rozšíření znamená, že při evoluci sítě není nutné vždy provádět dealokaci existující instance formuláře zpráv a jeho nahrazení instancí jinou.

V následujícím textu označíme ${}^i s$ jako i-tý znak řetězce s.

Pro identifikátory proměnných a konstant platí, že při provedení přechodu t je instance značky odebrána z místa $p \in P$: $N(a) = (p,t)$, $a \in A$ a následně je její instance nebo její identická kopie vložena do místa $q \in P$: $N(a) = (t, q)$, $a \in A$ pokud:

- Instance značky je odebrána z místa p prostřednictvím konstanty nebo proměnné v hranovém výrazu hrany (p,t). Identifikátor proměnné nebo konstanty nazvěme *input*.
- Instance značky je přidána do místa q prostřednictvím konstanty nebo proměnné v hranovém výrazu hrany (t,q). Identifikátor proměnné nebo konstanty nazvěme *output*.
- Identifikátory mají stejný první znak, tedy ${}^1 input = {}^1 output$.

Pro identifikátory proměnných v hranových výrazech platí, že jeho druhý znak slouží k určení obsahu *j*-té položky formuláře zpráv. Tohoto je využito zejména v rozhodovacích přechodech. Provedení rozhodovacího přechodu vypadá následovně:

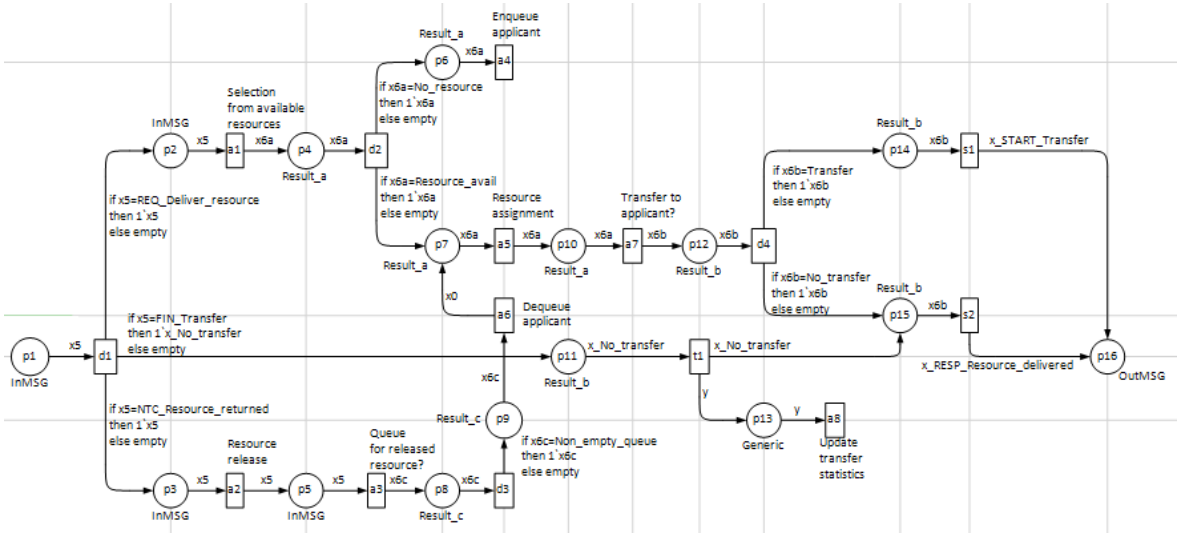
- instance značky je odebrána z místa $p \in P$: $N(a) = (p,t)$, $a \in A$ prostřednictvím proměnné v hranovém výrazu hrany (p,t). Identifikátor proměnné nazvěme *input*.
- Stanoví se hodnota $j = int(^{2}input)$. Hodnota *j* je tedy číslo z intervalu <0;9> získané transformací druhého znaku řetězce identifikátoru.
- Datový obsah *j*-té položky značky je pak použit jako konstanta pro vyhodnocování jednotlivých rozhodovacích hran, které z přechodu *t* vycházejí.

Třetí znak identifikátorů se využívá pouze pokud existuje více identifikátorů se shodnými prvními dvěma znaky, ale které jsou spojeny se značkami různých barev.

3.4 Stavový prostor Petriho sítí

Stavový prostor Petriho sítě představuje množinu značení dosažitelných ze značení počátečního. Lze jej vizualizovat pomocí grafu. Značení, také nazývané stav, tvoří vrcholy grafu, kroky tvoří hrany grafu. Pro popis můžeme značení rozepsat jako stavový vektor. Prvky stavového vektoru jsou místa sítě a hodnoty jsou multimnožiny barev značek, které se na místě vyskytují. Stavový prostor sítě je po vizualizaci vhodný pro analýzu sítě. Lze pomocí něj objevit problémy a chyby v návrhu.

Pro sítě ABA-CPN lze ve vrcholech grafu stavového prostoru sledovat výskyty značek v místech, na hranách grafu stavového prostoru pak lze sledovat provedený přechod, přenesené barvy pomocí proměnných v hranových výrazech hran sítě s přechodem incidentními a instance značek.



Obrázek 11 – Příklad sítě ABA-CPN

Stavový prostor této podtřídy sítí pro každé přípustné inicializační značení vždy:

- obsahuje alespoň jedno mrtvé značení,
- obsahuje přechod, který je živý,
- je konečný.

Přechod, který je živý pro každé přípustné inicializační značení, neboli který je pro každé přípustné inicializační značení proveden, je vstupní přechod.

Následuje popis příkladu stavového prostoru sítě ABA-CPN. Síť je zobrazena na obrázku 11 a stavový prostor na obrázku 12. Stavové vektory jsou pak rozepsány na obrázku 13 pro jedno ze tří možných počátečních značení. Na tuto síť se dále budeme odvolávat také v kapitole 4.4.5 . Pro síť jsou definovány následující množiny barev:

- *InMSG*={*REQ_Deliver_resource*, *NTC_Resource_returned*, *FIN_Transfer*},
- *OutMSG*={*RESP_Resource_delivered*, *START_Transfer*},
- *Result_a*={*Resoure_avail*, *NoResource*},
- *Result_b*={*Transfer*, *No_transfer*},
- *Result_c*={*Empty_queue*, N*on_empty_queue*},
- *Generic*={*e*},

následující konstanty:

- *x0=Resource_avail*,
- *x_RESP_Resource_delivered=RESP_Resource_delivered*,
- *x_START_Transfer=START_Transfer*,
- *x_No_transfer=No_transfer,*

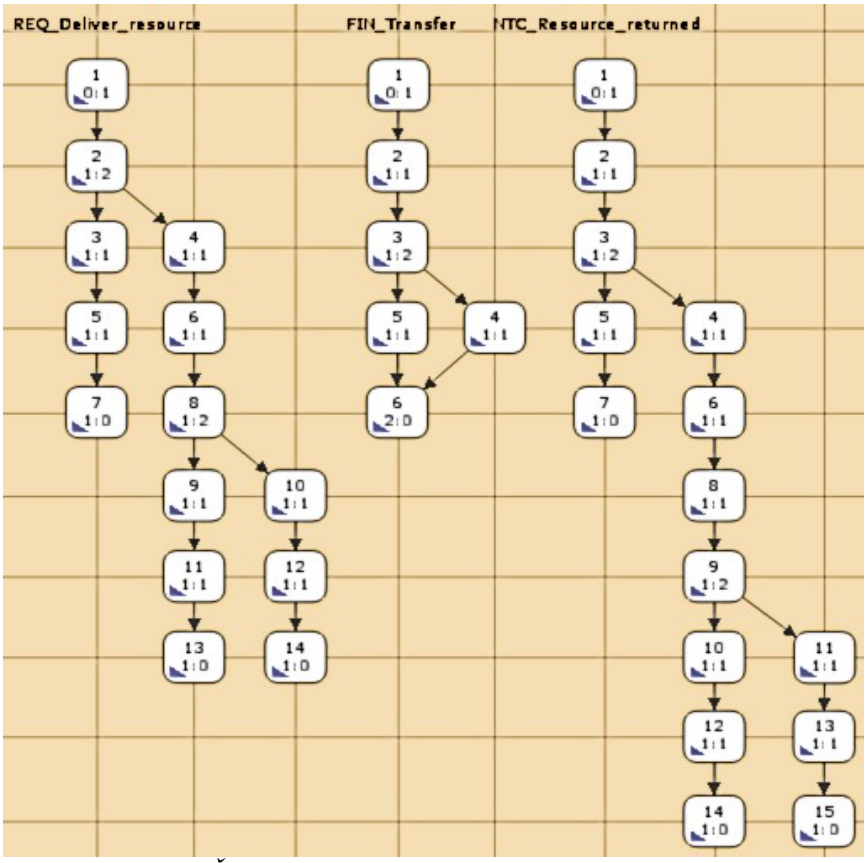
a následující proměnné:

- *y* : *Generic*,
- *x5* : *InMSG*,
- *x6a* : *Result_a*,
- *x6b* : *Result_b*,
- *x6c* : *Result_c*.

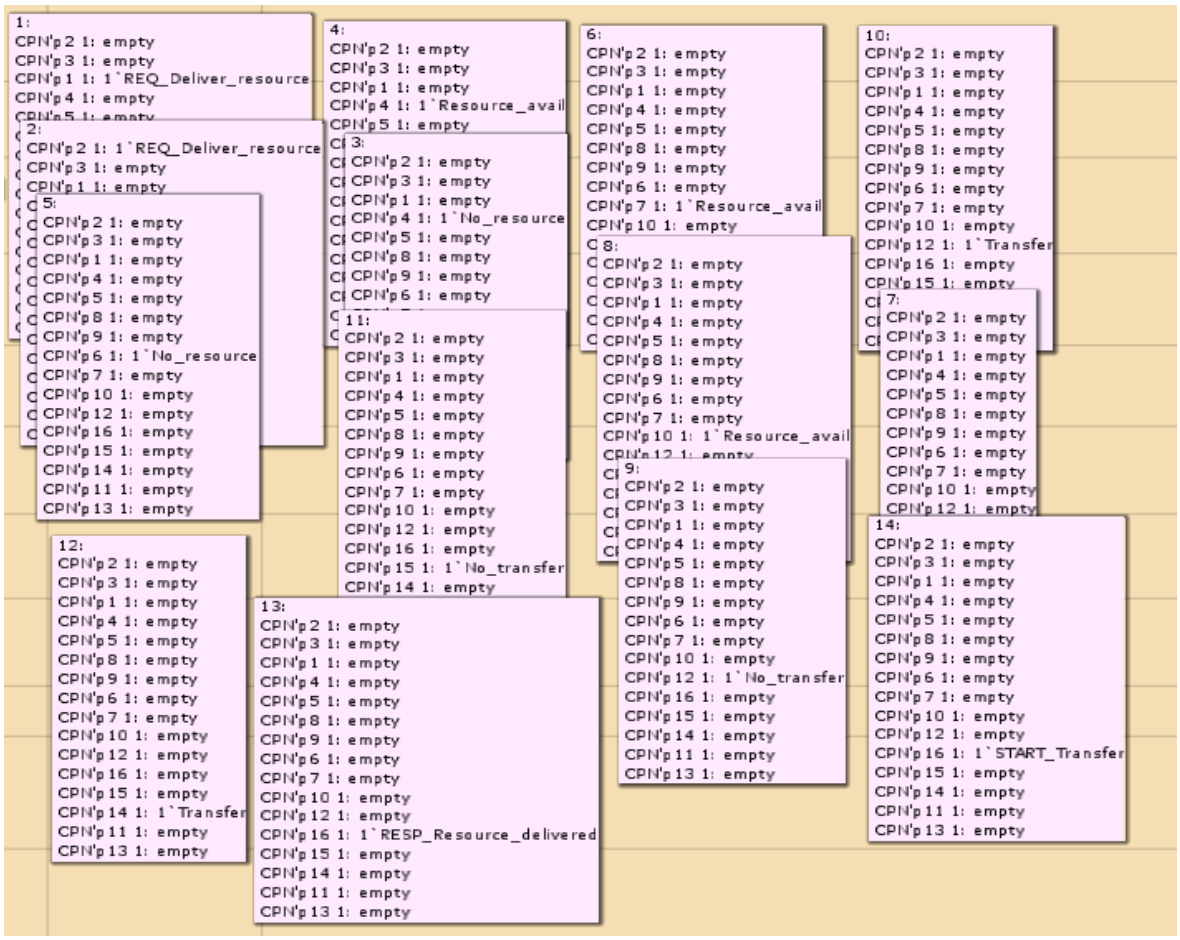
Stavový prostor je vypočítán v nástroji CPN Tools. Tento nástroj umožňuje výpočet grafu pouze pro jedno počáteční značení. Při výpočtu pro jiné počáteční značení nepřihlíží k předem vypočítaným grafům, takže není na první pohled patrné, že některé větve grafů jsou společné. Toto je omezení nástroje, jelikož je navržen pro práci s obecnými barvenými Petriho sítěmi, jejichž stavové prostory mohou být neomezené a s neomezenou množinou

počátečních značení. Jak již ale bylo řečeno výše, stavový prostor sítí ABA-CPN je vždy omezený, stejně jako množina počátečních značení sítě.

Aby bylo možné vystihnout tuto skutečnost a vypočítat kompletní graf stavového prostoru sítě ABA-CPN, byl v rámci této práce vyvinut nástroj, který je popsán v následujících kapitolách. Kompletní graf stavového prostoru sítě je zobrazen v kapitole 4.4.5 o testování programu a v příloze A.



Obrázek 12 – Částečné grafy stavového prostoru sítě ABA-CPN



Obrázek 13 – Stavové vektory stavů grafu vypočítaného stavového prostoru pro vstupní zprávu REQ_Deliver_resource

4 Popis aplikace

V této kapitole jsem popsal řešení mé aplikace pro výpočet stavového prostoru sítě ABA-CPN. Zahrnuje popis toho, co aplikace musí umět, popis datových struktur, které jsou při řešení použity, metod, které tvoří jádro aplikace a metod, které slouží k obsluze.

4.1 Požadavky na aplikaci

Pro to, aby aplikace mohla být prohlášena za funkční, musí splňovat následující hlavní požadavky:

- Musí být schopná vypočítat kompletní stavový prostor sítě ABA-CPN. To znamená, že pro každé přípustné inicializační značení musí umět najít všechna značení z něj dosažitelná.
- Musí být schopná vypočítaný stavový prostor vhodně zobrazit ve formě grafu.
- Musí být schopná vygenerovat textovou zprávu, v níž je přehled údajů o stavovém prostoru.

Pod pojmem kompletní stavový prostor je myšleno vypočítat stavový prostor najednou pro celou množinu všech přípustných počátečních značení M_0. Toto chování je odlišné např. od nástroje CPN Tools, které počítá stavový prostor vždy pouze pro aktuálně vybrané počáteční značení.

Další požadavky vyplynuly mj. z představ vedoucího práce. Tyto požadavky nejsou kritické pro funkci aplikace, ale poskytují uživatelský komfort:

- Graf stavového prostoru musí jít po grafické stránce upravovat. Způsoby úpravy by měly být co nejjednodušší a co nejméně pracné.
- Jak vrcholy, tak hrany grafu musí mít popis, který popisuje konkrétní stav respektive konkrétní informace o provedení přechodu.
- Aplikace musí umožňovat ukládání a nahrávání už vypočítaných grafů.
- Aplikace musí umožňovat výstup grafu pro prohlížení mimo aplikaci, nejlépe jeho tisk nebo export do PDF.
- Aplikaci by mělo být možno propojit s editorem sítí ABA-CPN, který vyvíjí Bc. Petr Nejman.

Programovací jazyk mohl být libovolný, avšak z důvodu možnosti propojení s editorem jsem vybral jazyk C# a platformu Microsoft .NET, jelikož editor již byl v tomto jazyku rozpracován.

4.2 Popis použitých datových struktur

V této podkapitole popisuji datové struktury, se kterými v programu pracuji. U každé struktury popisuji jak funguje, jak je implementována, operace, které využívám a složitosti těchto operací, pokud jsou pro danou implementaci známé. Ve zbytku kapitoly 4 a v kapitole 5 , při popisu algoritmů a funkcí programu, pak uvádím konkrétní použití struktur a důvody, proč jsem je vybral. Proto příklady použití v této podkapitole neuvádím.

Následující struktury jsou seřazeny od těch, které již jsou implementovány v programovacím jazyku po ty, které jsem implementoval dodatečně. Pro všechny použité struktury jsou použity generické varianty.

4.2.1 Fronta

Fronta je lineární datová struktura. Uchovává prvky v pořadí, ve kterém do ní byly vloženy. Prvky jsou vždy vkládány na konec fronty a odebírány z jejího počátku. V programovacím jazyku C# ji reprezentuje třída Queue, založená na lineárním seznamu na poli.

Operace, které fronta umožňuje, jsou následující:

- Přidání prvku na konec fronty. Tato operace má obvykle složitost O(1). Nejhorší složitost je O(n), je-li potřeba alokovat více místa pro vnitřní pole, ve kterém jsou prvky uchovávány.
- Odebrání prvku ze začátku fronty. Tato operace má vždy složitost O(1).
- Test, je-li fronta prázdná. Tato operace má vždy složitost O(1).

4.2.2 Množina

Množina je lineární datová struktura. Uchovává prvky, které jsou samy sobě klíčem. Nepovoluje duplicitní klíče. V programovacím jazyku C# jí reprezentují třídy Implementující rozhraní ISet, tedy třídy HashSet a SortedSet. V programu používám pouze třídu SortedSet, proto při popisu operací jsou složitosti uvedeny pouze pro ni.

Operace, které množina umožňuje, jsou následující:

- Přidání prvku do množiny. Pro tuto operaci složitost nebyla uvedena.
- Odebrání prvku z množiny. Tato operace má vždy složitost O(1).
- Zjištění počtu prvků v množině. Tato operace má vždy složitost O(1).
- Zjištění, zda množina obsahuje prvek. Tato operace má vždy složitost O(log n).
- Prohlídka všech prvků množiny. Tato operace má vždy složitost O(n).

4.2.3 Lineární seznam

Lineární seznam je lineární datová struktura, která uchovává prvky. Nepracuje s klíči. K prvkům můžeme přistupovat, pokud známe jejich pozici v seznamu. Obecně je neutříděný. V jazyku C# jej reprezentují třídy implementující rozhraní IList, tedy třídy List (seznam na poli) a LinkedList (dynamický zřetězený seznam). Ve své aplikaci používám seznam implementovaný třídou List.

Operace, které seznam umožňuje jsou následující:

- Přidání prvku na konec seznamu. Tato operace má obvykle složitost O(1). V nejhorším případě je složitost O(n), je-li potřeba alokovat více místa pro vnitřní pole, ve kterém jsou prvky uchovávány.
- Odebrání prvku ze seznamu. Tato operace má vždy složitost O(n),
- Nalezení pozice prvku. Tato operace má vždy složitost O(n).
- Přístup k prvku na n-té pozici. Tato operace má vždy složitost O(1).
- Zjištění počtu prvků v seznamu. Tato operace má vždy složitost O(1).
- Prohlídka všech prvků seznamu. Tato operace má vždy složitost O(n).

Za zmínku stojí, že ačkoliv třída List implementuje pro třídění metodu Sort(), rozhraní IList, pomocí kterého k této struktuře často přistupuji, tuto metodu nespecifikuje. Pro třídění prvků je tedy potřeba přetypování. Pokud nelze objekt seznamu přetypovat na třídu List, je potřeba provést export prvků do pole, které je následně setříděno (složitost O(n ln n), v nejhorším případě $O(n^2)$) a znovu vloženo do seznamu.

4.2.4 Tabulka

Tabulka je lineární datová struktura, může však být založena i na datové struktuře hierarchické. Uchovává prvky a umožňuje k nim přístup na základě klíče. V programovacím jazyku C# ji reprezentují třídy implementující rozhraní IDictionary, tedy třídy SortedList, SortedDictionary a Dictionary, přičemž já používám převážně Dictionary, tabulku založenou na tabulce hashovací. Pro případy, kdy potřebuji přístup i pomocí indexu, využívám třídu SortedList, tabulku založenou na dvojici setříděných seznamů na poli.

Operace, které tabulka umožňuje, jsou následující:

- Přidání nové dvojice klíč/hodnota. Pro Dictionary je složitost operace O(1), nebo O(n), je-li potřeba alokovat více místa pro vnitřní pole, ve kterém jsou prvky uchovávány. Pro SortedList je složitost O(log n), není-li potřeba alokovat více místa a zároveň je prvek vložen na konec pole, jinak je složitost O(n).
- Odebrání dvojice klíč/hodnota. Pro Dictionary je složitost operace O(1). Pro SortedList je složitost O(n).

- Přístup k hodnotě pomocí klíče. Pro Dictionary je složitost operace O(1). Pro SortedList je složitost O(log n), pokud se pouze čte nebo mění hodnota. Pokud se mění klíč, je složitost O(n).
- Prohlídka všech hodnot nebo klíčů v tabulce. Pro obě třídy je složitost operace O(n).
- Získání počtu dvojic klíč/hodnota. Pro obě třídy je složitost operace O(1).
- přístup ke klíči nebo hodnotě podle indexu. Tuto metodu umožňuje pouze třída SortedList, a to se složitostí O(1).

4.2.5 2D Strom

2D strom je stromová struktura. Uchovává prvky na základě dvojice klíčů. Jeho použití je výhodné při uchovávání dvojrozměrných dat, např. grafických primitiv ve 2D prostoru. Pro aplikaci jsem ji musel doprogramovat. Zastřešuje jej rozhraní ID2Tree, které implementuje třída D2Tree. Strom je vystavěn na základě binárního stromu, čímž je umožněno rychlé vyhledávání v rozsahu klíčů.

Operace, které 2D strom umožňuje, jsou následující:

- Přidání prvku. Tato operace má složitost O(log n).
- Odebrání prvku. Tato operace má složitost odhadem $O(\log^2 n)$.
- Vyhledání prvku. Tato operace má složitost O(log n).
- Vyhledání prvků v rozsahu klíčů. Tato operace má složitost O(log n).
- Prohlídka všech prvků stromu. Tato operace má složitost O(n).

Složitost operací, které vyžadují traverz struktury, tedy přidání, odebrání a vyhledání prvku, může dosáhnout v nejhorším případě O(n), pokud strom zdegeneruje do lineární struktury. Existuje možnost, že by se strom přebudoval při každé změně, ale operace přebudování je velmi náročná, její složitost je odhadem $O(n*\log^2 n)$. Druhá možnost je zjišťovat, jak moc je strom nevyvážený, ale nepřišel jsem na vhodný způsob, jakým tento údaj zjišťovat. Proto neexistuje metoda, která by strom přebudovala. Strom se snažím vyvažovat už při jeho plnění (viz kapitoly 4.4.3 a 4.5.5). Přesuny prvků ve struktuře, které se ději při změnách jejich polohy dále neřeším.

4.2.6 Orientovaný graf

Orientovaný graf je kompozitní struktura. Skládá se z vrcholů a hran, přičemž hrany mohou být orientované nebo neorientované, a oba typy prvků mohou být ohodnocené nebo neohodnocené. Pro aplikaci jsem graf musel doprogramovat. Jeden typ grafu, který reprezentuje graf sítě CPN, je implementován na základě rozhraní ICPN, pomocí hashovacích tabulek uchovávajících vrcholy a hrany ve třídě CPNet na základě jejich identifikátoru. Druhý typ grafu, který reprezentuje graf stavového prostoru, nemá

samostatné zastřešovací rozhraní a je implementován pomocí 2D stromu pro uchovávání vrcholů a seznamem pro uchovávání samostatných referencí na hrany.

Operace, které graf umožňuje jsou:

- přidání vrcholu grafu,
- přidání hrany grafu,
- odebrání vrcholu grafu,
- odebrání hrany grafu,
- přístup k vrcholu grafu,
- přístup k hraně grafu.

Pro graf stavového prostoru platí složitosti pro vkládání, odebírání a vyhledávání vrcholů jako pro operace vkládání, odebírání a vyhledávání u 2D stromu, pro hrany jako u lineárního seznamu. Ke hranám se v tomto případě obvykle přistupuje pouze v rámci procházení celé kolekce nebo při procházení hran incidentních k určitému vrcholu.

Pro graf sítě jsou tyto operace konstantní, přičemž každý vrchol a každá hrana je identifikována jedinečným číslem.

Vyhledání incidentních hran k vrcholům a incidentních vrcholů k hranám jsou pro oba typy grafů konstantní, jelikož si na sebe navzájem uchovávají reference.

4.3 Popis grafické reprezentace stavového prostoru

Prvky grafu stavového prostoru jsou do značné míry inspirovány aplikací CPN Tools. Vrcholy jsou reprezentovány obdélníky s číslem stavu, přičemž pro každý lze zobrazit v samostatném prvku výpis stavového vektoru. Pro každou hranu lze také zobrazit informace o změně.

V původním návrhu vizualizace byla pozice těchto dodatečných oken pevně spjata s rodičovským prvkem. Na základě požadavku a hlubšího zamyšlení jsou však i tyto prvky oddělitelné a samostatně pohyblivé. Důvod je přehlednost, kdy je možno zobrazit několik takovýchto dodatečných oken, aniž by překrývaly jiné prvky grafu. Oproti CPN Tools však po oddělení zůstává jejich pozice nezávislá na změně pozice rodičovského prvku. Tímto je možné upravovat graf i po tom, co jsou všechny tyto prvky umístěny okolo samotného grafu a jejich pohyb není žádoucí. Pro tyto prvky lze vždy provést zpětné napojení na kotvící místa. Po napojení na tato kotvící místa jejich pozice nebude jejich pozice nezávislá.

Žádný z prvků není samostatně roztažitelný, jelikož mi to přišlo zbytečné. Vrcholy grafu jsou dostatečně široké, aby obsáhly čtyřmístné číslo stavu a popisy stavových změn jsou vždy tak veliké, aby se do nich vešel celý jejich text. Pro výpisy stavových vektorů by měnitelná velikost měla smysl, jelikož tyto výpisy mohou zabírat mnoho řádků. Pro jednu síť tyto výpisy ale budou mít text porovnatelně dlouhý a proto dává větší smysl ponechat

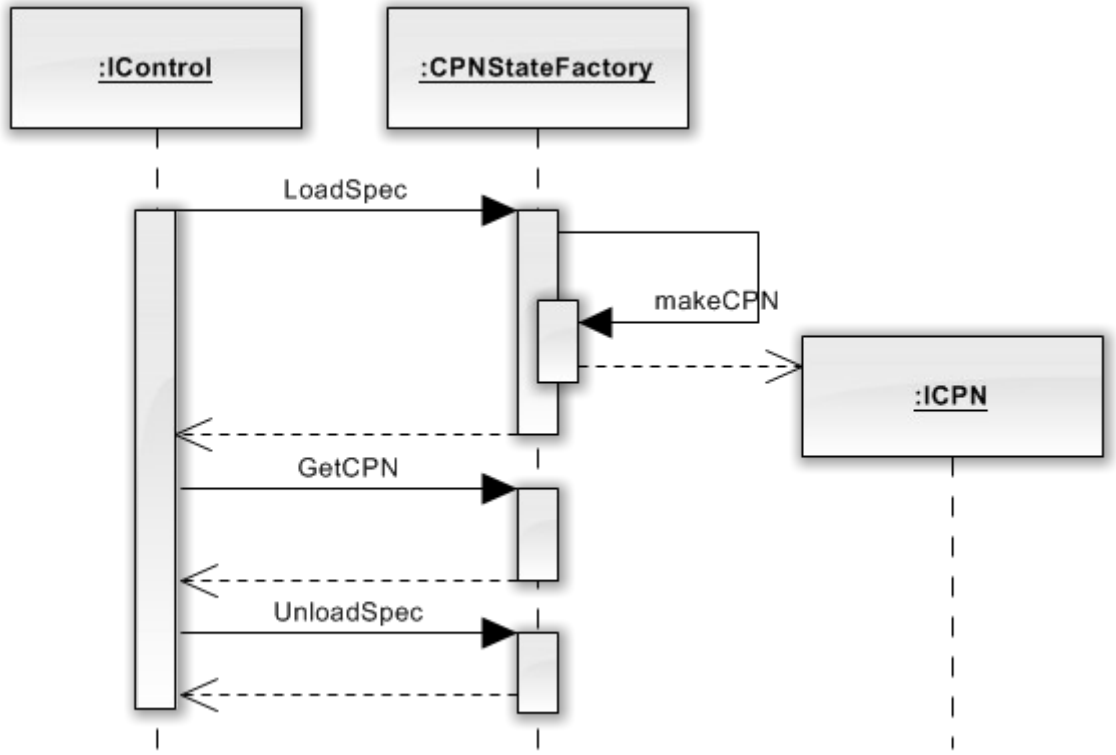
nastavení globální a případné přetečení kompenzovat možností posunování zobrazených řádků.

Graf stavového prostoru se v mojí aplikaci na rozdíl od CPN Tools vykresluje najednou a jsou zaručeny alespoň minimální odstupy jednotlivých vrcholů po generování. Díky tomu je graf relativně přehledný i bez dodatečných úprav. Protože jsou u každého vrcholu také implicitně vidět všichni předchůdci i následníci, nezobrazuje se informace o jejich počtu.

4.4 Popis podstatných tříd a metod pro výpočet stavového prostoru

4.4.1 Vytvoření sítě

Pro výpočet stavového prostoru sítě CPN je potřeba, aby tato síť existovala. Část programu, která se o tvorbu stará, je navržena podle návrhového vzoru abstraktní továrna. O tvorbu sítě se stará třída implementující rozhraní ICPNFactory, která navrací objekty implementující rozhraní ICPN a další, které představují dodatečné informace pro síť podstatné. Ve výchozím případě je tovární třída CPNStateFactory.



Obrázek 14 – Sekvenční diagram tvorby sítě

Obrázek 14 popisuje operace, které se vykonávají při vytváření sítě. Pro zjednodušení je uvedena tvorba pouze sítě jako takové. Ve skutečnosti se tvoří ještě kolekce množin barev, kolekce konstant a kolekce proměnných, které se v síti používají.

Při tvorbě sítě se vychází z konvencí pojmenovávání sítí ABA-CPN uvedených v kapitole 3.3.2 , tedy:

- vstupní místo je místo označené jako p_1,
- výstupní místo je označené jako p_n, kde $n = |P|$,
- rozhodovací přechod má označení, jehož první znak je písmeno d,
- asistenční přechod má označení, jehož první znak je písmeno a,
- odesílací přechod má označení, jehož první znak je písmeno s,
- standardní přechod má označení, jehož první znak je písmeno t.

Tvorba sítě probíhá na základě dodaného XML dokumentu, který obsahuje předpis sítě ve formátu stejném, jako používá nástroj CPN Tools. Všechny součásti, které jsou pro výpočet stavového prostoru potřeba, jsou vypočítány najednou, jelikož spolu souvisí. Souvislost je výrazná hlavně u hran, kde na základě jejich hranových výrazů třída CPNStateFactory rozhoduje, zda je hrana elementární, konstantní nebo rozhodovací.

4.4.2 Verifikace sítě

Po vytvoření sítě je potřeba provést její verifikaci, tedy zjistit, zda její struktura odpovídá struktuře sítě ABA-CPN. Pro tento účel jsem s povolením od Bc. Petra Nejmana použil jeho upravenou verifikační třídu. Tato statická třída obsahuje metody, které zaznamenají odchylky od korektní struktury, což jsou:

- duplicitní názvy v deklaraci,
- prázdné množiny barev,
- deklarace, které obsahují jedno z klíčových slov,
- neexistence vstupního místa,
- existence více než jednoho vstupního místa,
- neexistence výstupního místa,
- existence více než jednoho výstupního místa,
- existence míst, která nemají přidělenu množinu barev,
- existence míst, která mají přidělenu neznámou množinu barev,
- existence ne-vstupních míst, do kterých neústí žádná hrana,
- existence hran, které ústí do vstupního místa
- existence ne-výstupních míst, ze kterých neústí žádná hrana,
- existence hran, které ústí z výstupního místa
- existence přechodů, do nichž neústí žádná hrana,

- existence rozhodovacích přechodů, z nichž nevychází jen rozhodovací hrany,
- existence ne-standardních přechodů, do nichž ústí více jak jedna hrana,
- existence rozhodovacích přechodů, z nichž nevychází žádná hrana,
- existence odesílacích přechodů, z nichž nevychází žádná hrana,
- existence hran, které nedisponují žádným výrazem,
- existence elementárních hran, jejichž výraz nabývá neznámé hodnoty,
- existence elementárních hran, jejichž výraz obsahuje proměnnou, která nespadá pod množinu barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence konstantních hran, jejichž výraz nabývá neznámé hodnoty,
- existence konstantních hran, jejichž výraz obsahuje konstantu, která nespadá pod množinu barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence konstantních hran, jejichž výraz obsahuje barvu, která nespadá pod množinu barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, které nevychází z rozhodovacího přechodu,
- existence rozhodovacích hran, jejichž výraz neobsahuje proměnnou ze vstupu incidentního přechodu,
- existence rozhodovacích hran, jejichž výraz obsahuje nesplnitelnou podmínku,
- existence rozhodovacích hran, jejichž výraz nabývá pro "*then*" i "*else*" hodnoty empty,
- existence rozhodovacích hran, jejichž výraz nabývá pro "*then*" proměnné, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*then*" konstanty, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*then*" barvy, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*then*" neznámé hodnoty,

- existence rozhodovacích hran, jejichž výraz nabývá pro "*else*" proměnné, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*else*" konstanty, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*else*" barvy, která nespadá do množiny barev incidentního místa (pokud je množina v tomto místě definovaná),
- existence rozhodovacích hran, jejichž výraz nabývá pro "*else*" neznámé hodnoty,
- existence cyklu.

Oproti původní třídě Bc. Nejmana byly vypuštěny testy inicializačního značení. Při výpočtu stavového prostoru mým programem neberou v potaz existující značky. Všechna přípustná počáteční značení jsou automaticky generována aplikací.

Pokud je při verifikaci odhalena některá z výše uvedených chyb, výpočet stavového prostoru neproběhne a je zobrazena chybová hláška.

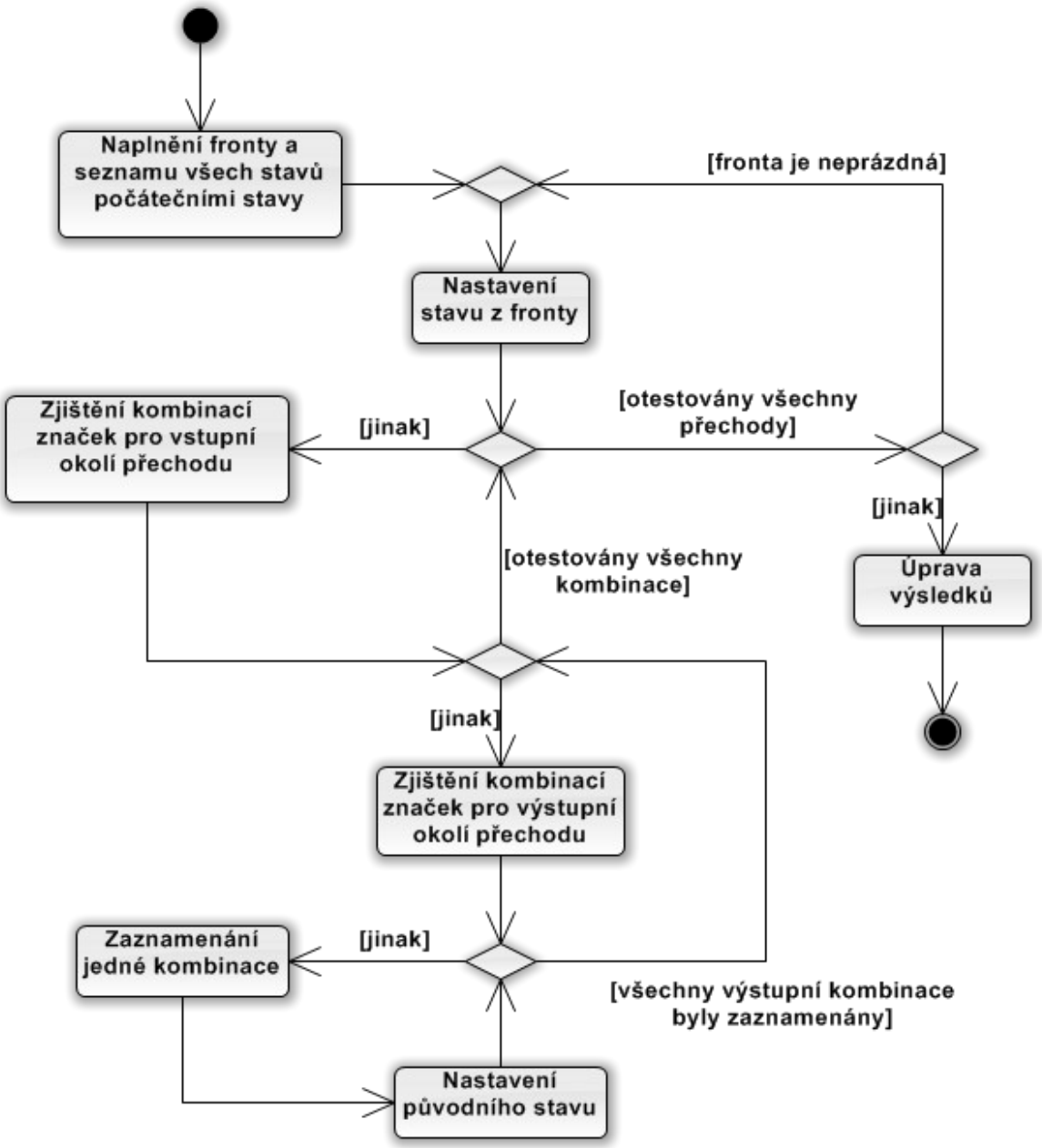
4.4.3 Výpočet stavového prostoru

Funkce výpočtu stavového prostoru je pro aplikaci nosná funkce. Spouští se ihned po načtení sítě.

Algoritmus pro výpočet zobrazuje obrázek 15. Funkce je rozdělena do několika kroků.

Krok 1: V tomto kroku dojde k inicializaci výpočetní funkce, při které se z každého místa sítě odeberou značky, které se na nich vyskytují. Následně dojde k naplnění fronty, která eviduje stavy, k němž se ještě nehledaly stavy následující. Fronta se naplní všemi počátečními stavy. Počáteční stav se vytvoří tak, že se na vstupní místo přidá značka s barvou z $C(p_1)$. Počáteční stavy se zároveň uloží do hashovací tabulky, která eviduje stavy, kterých již při výpočtu bylo dosaženo, a jim odpovídající vrcholy grafu stavového prostoru. Dále se pokračuje krokem 2.

Použití hashovací tabulky je pro uchovávání vypočítaných stavů výhodné z toho důvodu, že umožňuje rychlé vyhledávání již vypočítaných stavů. Při výpočtu je potřeba poznat, zda jsme se nenapojili na větev grafu stavového prostoru, které jsme předtím dosáhli z jiného stavu. Stavové vektory, kterými jsou stavy definované, obsahují tolik prvků, kolik má síť míst, a na každém místě se může teoreticky nacházet neomezený počet značek. Jelikož jsou značky uchovávány jako řetězce, trvalo by postupné porovnávání velmi dlouho, stejně jako porovnávání řetězců, které můžeme ze stavových vektorů sestavit. Možnost výpočtu hashe se jeví jako rychlá a jednorázová alternativa a hashovací tabulky pro vyhledávání i pro vkládání prvků poskytují konstantní složitost.



Obrázek 15 – Hlavní algoritmus výpočtu

Krok 2: Z fronty se vybere stav a podle něj se nastaví značení sítě. Dále se načte seznam přechodů sítě a pro každý přechod z tohoto seznamu se provádí krok 3. Pokud je fronta prázdná, výpočet končí a obnoví se značení, které měla síť před začátkem výpočtu.

Krok 3: Otestuje se přechod. První věc, která se u přechodu zjišťuje, je, zda mají všechna místa v jeho vstupním okolí alespoň jednu značku. Pro většinu přechodů to znamená místo jedno, ovšem kvůli specifikaci přechodů standardních je jejich počet obecně větší než 0. Je-li podmínka splněna, přistupuje se k vytváření kombinací značek, které mohou projít. Ve svém řešení vytvořím samostatné kolekce pro jednotlivá místa, které následně procházím pomocí enumerátorů v kroku 4.

Jakmile je krok 3 dokončen, opakuje se pro následující přechod, nebo se pokračuje krokem 2, pokud byly otestovány všechny přechody.

Krok 4: Pro možnou kombinaci značek přenášených ze vstupu zkouším, zda:

- danou barvu značky odpovídající hrana propustí,
- mají hrany se stejnou proměnnou ve hranovém výrazu stejnou barvu značky.

Pokud alespoň jedna z těchto podmínek neplatí, opakuje se krok 4 pro následující vstupní kombinaci, nebo krok 3, pokud další vstupní kombinace neexistuje.

Pokud obě podmínky platí, je daná vstupní kombinace průchozí, a je možné pro ni odvodit výstupní kombinace značek. V mém řešení vytvářím kolekce možných značek pro každé místo ve výstupním okolí přechodu.

Přitom je potřeba dodržovat následující zásady:

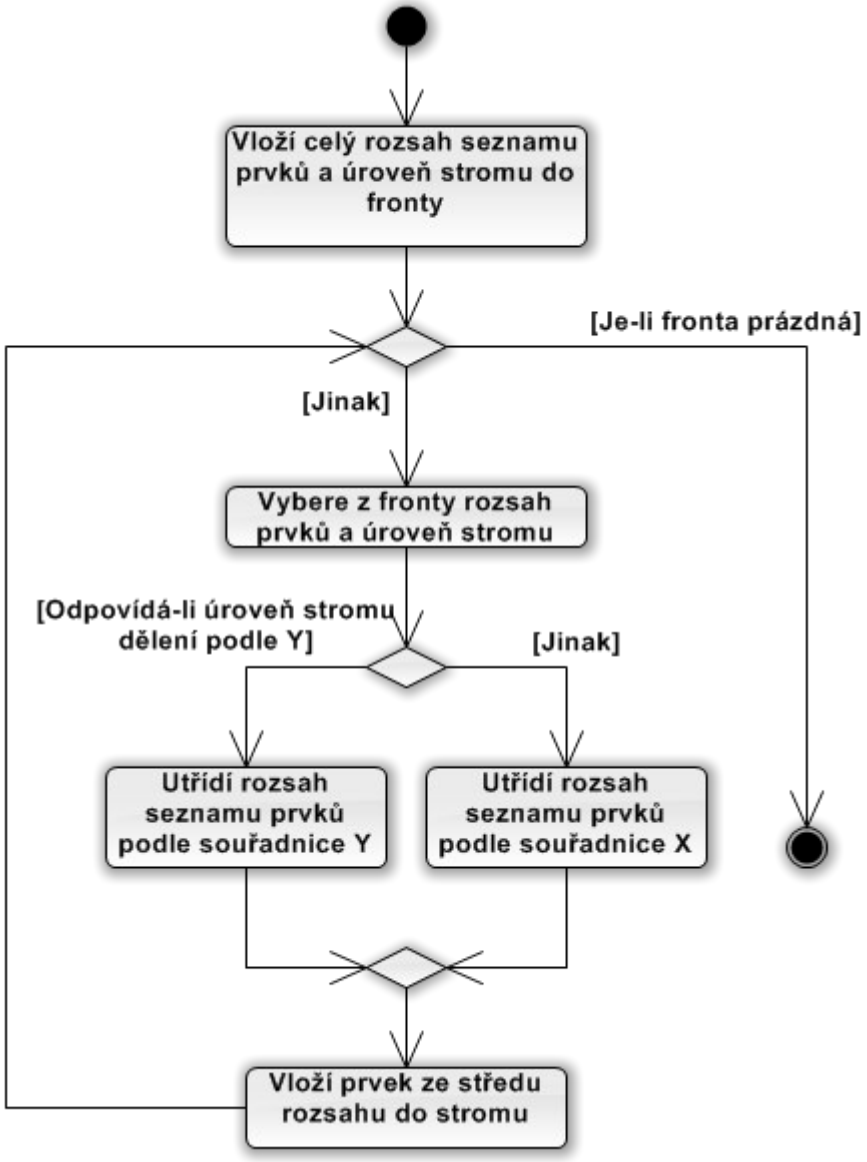
- pokud je výstupní proměnná stejná jako vstupní proměnná, musí se barvy odebrané a přidané značky shodovat,
- pokud jsou dvě výstupní proměnné stejné, barva vkládaných značek se musí shodovat,
- pokud je barva vstupního místa stejná jako barva výstupního místa a identifikátor proměnné ve hranovém výrazu výstupní hrany má stejný prefix (první znak) jako identifikátor konstanty nebo proměnné hranového výrazu vstupní hrany, musí se barvy odebrané a přidané značky shodovat,
- pokud je barva výstupního místa různá od barvy vstupního místa a/nebo proměnná nemá stejný prefix, může značka nabývat kterékoliv barvy z množiny barev místa.

Při vytváření kolekcí značek také zjišťuji, jak se mění instance formuláře, které značky přenáší. V tomto případě opět záleží na prefixu proměnných a konstant ve hranových výrazech. Jsou-li prefixy shodné, instance se zachovává. Pokud jsou prefixy různé, dochází k alokaci instance nové. Při provedení přechodu tedy mohou nastat tři různé situace:

- původní instance formuláře pokračuje nezměněna a k alokaci instance nové nedošlo,
- původní instance formuláře pokračuje nezměněna, ale paralelně došlo k alokaci instance dodatečné,
- původní instance formuláře byla dealokována a došlo k alokaci instance nové.

Dále se pokračuje krokem 5. Pokud neexistuje další vstupní kombinace značek, pokračuje se krokem 3.

Krok 5: Upravuje se značení podle aktuální výstupní kombinace. Po každé změně je vytvořen stavový vektor a porovnán s již existujícími. Pokud nastane shoda, přidá se do grafu stavového prostoru hrana mezi vrcholy, které představují stavy předchozí a nalezený, jinak se vytvoří vrchol nový a hrana se vloží mezi něj a vrchol, který představuje stav předchozí. Pokud je stav nový, vloží se navíc do fronty stavů, ke kterým ještě nebyli hledáni následníci a do tabulky nalezených stavů. Následně je značení sítě obnoveno do předchozího stavu a opakuje se krok 5 pro následující výstupní kombinaci. Pokud žádná další výstupní kombinace neexistuje, pokračuje se krokem 4.



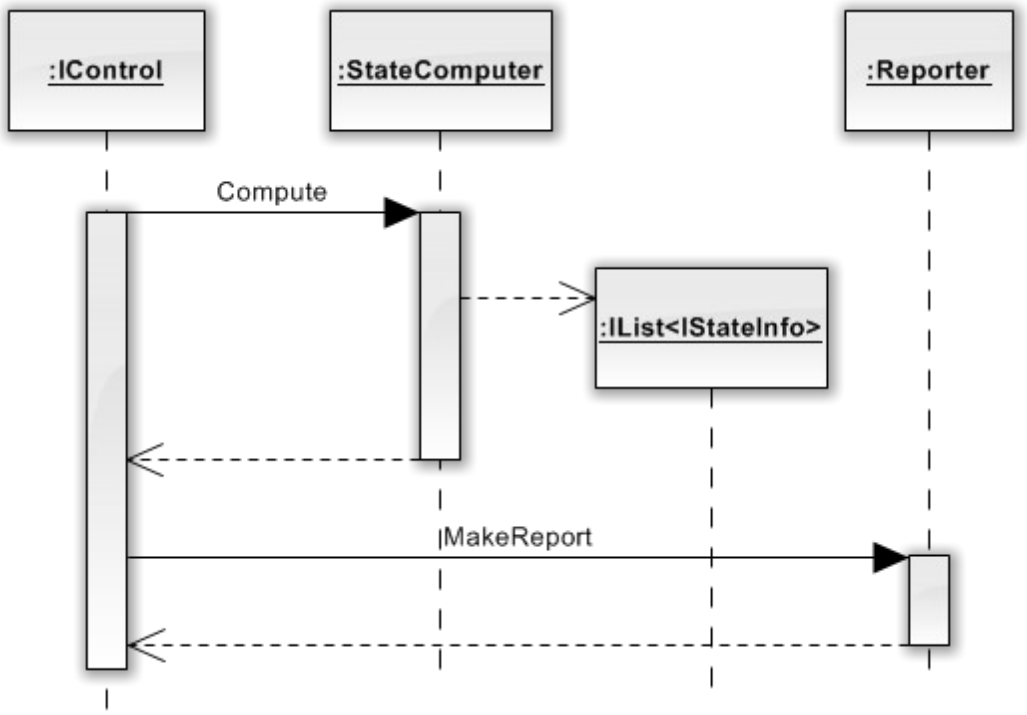
Obrázek 16 – Algoritmus vkládání prvků z lineárního seznamu do 2D stromu

Vypočítaný stavový prostor je potřeba ještě graficky zobrazit. Jednotlivé prvky jsou vytvořeny v ten okamžik, kdy jsou známy všechny údaje, které mají reprezentovat. To znamená, že vrchol grafu je vytvořen, jakmile je znám stavový vektor, který má představovat, a hrana grafu je vytvořena, jakmile je známá spojitost mezi dvěma vrcholy.

Dodatečné prvky grafu, tedy výpis stavového vektoru stavu a výpis informací o změně stavu jsou vytvořeny společně s odpovídajícím vrcholem, respektive hranou.

Pro grafické zobrazení je potřeba prvky rozmístit co nejpřehledněji na 2D kreslící plochu. Pro tento účel při výpočtu zároveň sleduji, kolik vrcholů grafu tvoří jednotlivé vrstvy. Jedna vrstva je tvořena všemi vrcholy, které jsou následníky vrstvy vyšší, přičemž nejvyšší vrstvu tvoří vrcholy představující výchozí stavy. Proto je při výpočtu využita fronta pro uchovávání stavů, ke kterým nebyli hledáni následníci. Zaručí, že stavy v pořadí, v jakém byly vytvořeny, odpovídají postupně jednotlivým vrstvám.

Vrcholy jedné vrstvy jsou vykresleny na jeden řádek. Pokud má vrchol následníky v další vrstvě, jsou vykresleny tak, aby byla horizontální souřadnice prvního následníka stejná jako horizontální souřadnice jeho předchůdce. Vertikální souřadnice je vždy větší, tzn. následník je umístěn pod předchůdce. Vertikální vzdálenost závisí na počtu prvků ve vrstvách následníka a předchůdce, přesněji ta vrstva, jež má více prvků, určuje vzdálenost. Za každý prvek je vzdálenost větší, do určité hranice, aby se graf zase neroztahoval příliš. Pokud má vrchol následníky v předchozí vrstvě, tak je jejich pozice už dána a automaticky se nemění.



Obrázek 17 – Sekvenční diagram výpočtu.

Všechny prvky grafu kromě hran jsou vkládány do 2D stromu, čímž je zaručeno, že pozice jejich levých horních rohů jsou vždy jedinečné. Tímto se zamezí situacím, kdy se překrývají dva prvky stejného typu, což je jedna z věcí, které nejvíce ovlivňují přehlednost. Aby se omezila degenerace struktury 2D stromu na strukturu lineární, je pořadí vkládání prvků závislé na jejich pozici. Nejprve je do stromu vložen prvek, jehož pozice je medián z pohledu horizontální souřadnice. Tento prvek také rozdělí celkovou kolekci prvků na dvě

poloviny, kde jedna polovina má souřadnici X větší a druhá menší nebo rovnou tomuto prvku. Takto se do stromu rekurzivně naskládají prvky, jejichž hodnota souřadnice je medián pro danou část kolekce. Souřadnice se na každé úrovni dělení střídají. Algoritmus vkládání do stromu je popsán na obrázku 16.

Kromě 2D stromu jsou vrcholy grafu uchovány i v lineárním seznamu, ve kterém jsou setříděny nejprve podle toho, zda jsou stav výchozí, obyčejný nebo terminální a pak podle jejich čísla. Tento seznam byl zaveden pro možnost vybírat vrcholy podle indexu v seznamu stavů v hlavním okně aplikace. Další využití našel pro funkce zpět a opakovat popsané v kapitole 4.5.6 .

4.4.4 Generování zprávy o stavovém prostoru

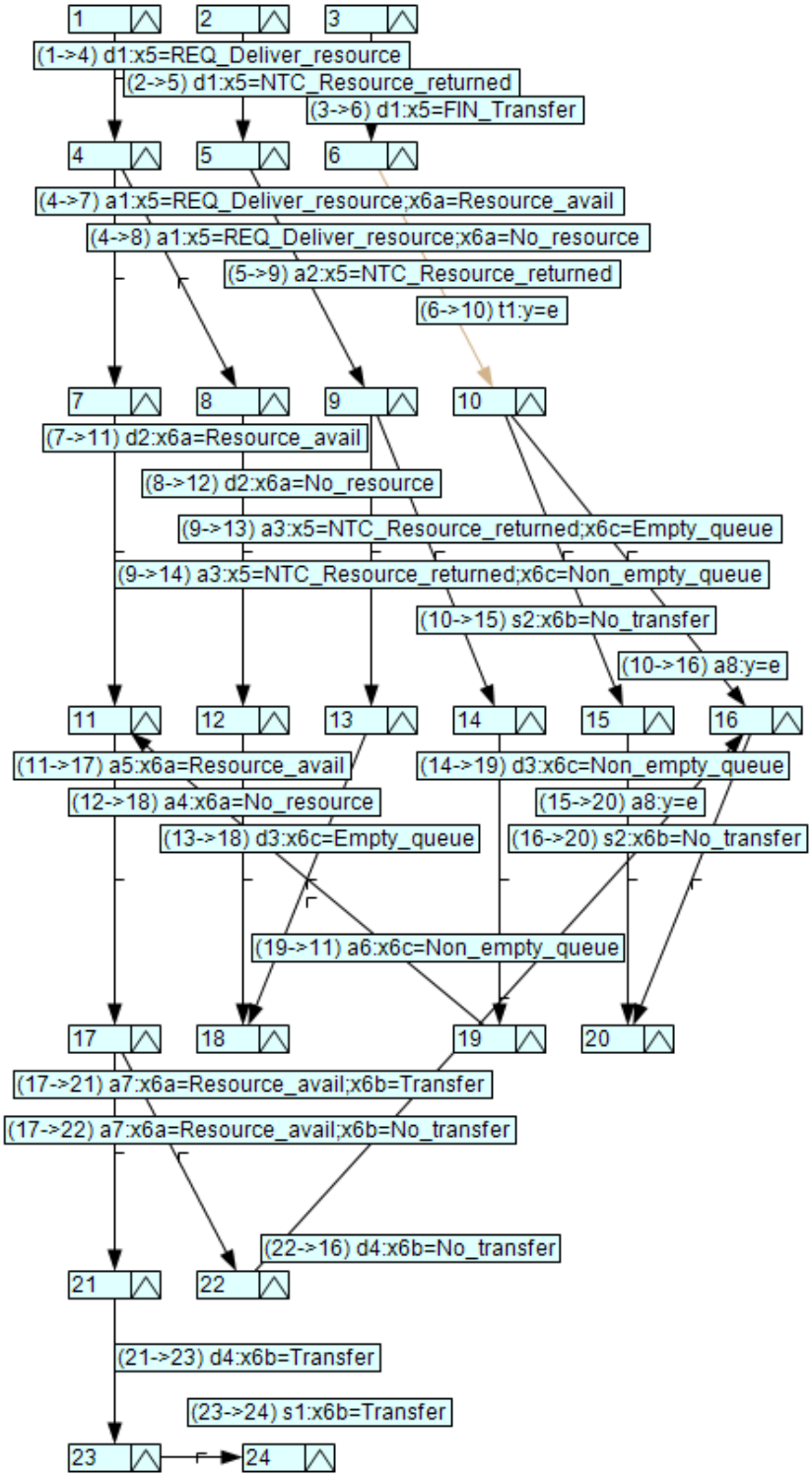
Jak je patrné na obrázku 17, generování zprávy je prováděno ihned po výpočtu stavového prostoru. Stará se o to zvláštní třída Reporter s jedinou metodou MakeReport.

Zobrazované informace jsou obdobné informacím generovaným v nástroji CPN Tools. Zahrnují:

- datum a čas výpočtu,
- celkový počet vrcholů stavového prostoru,
- celkový počet hran stavového prostoru,
- celkový počet počátečních stavů, neboli vrcholů stavového prostoru bez předchůdce,
- celkový počet terminálních stavů, neboli vrcholů stavového prostoru bez následníka,
- počet terminálních stavů, které neodpovídají korektnímu ukončení evoluce sítě ABA - CPN (popsané v kapitole 3.3.1),
- nejvyšší dosažené počty značek v místech a jejich hodnoty,
- seznam čísel stavů, které představují stavy terminální, včetně vyznačení nekorektních,
- seznam přechodů, které nebyly provedeny při žádné změně stavu sítě.

Tyto informace jsou nejprve sesbírány a až pak vypisovány. Metoda GetReport je navržena tak, aby dokázala plně využít průchodu kolekcemi vrcholů k sesbírání maxima údajů.

V inicializační fázi vynuluje čítače a naplní množinu přechodů všemi přechody, které se nachází v síti, ze které se stavový prostor počítá.



Obrázek 18 – Příklad kompletního stavového prostoru sítě

Při procházení vrcholů grafu testuje, zda reprezentuje stav počáteční, terminální, nebo nekorektní terminální. Následně spočítá, kolik se vyskytuje značek na každém místě, a pokud je pro některé počet značek větší než ve všech předchozích stavech, uloží si jejich

barvy. Je také potřeba dávat pozor na možnost, že počet značek nemusí být větší, ale stejný. V tomto případě se barvy značek také musí uložit, ale jako doplněk k už existujícímu seznamu barev značek. Také se prochází všechny hrany, které z vrcholu grafu vychází. Ze hran se vybere informace o provedeném přechodu. Provedený přechod se odebere z množiny všech přechodů. Tímto vznikne po průchodu kolekcí množina, která obsahuje pouze přechody, které nebyly provedeny.

4.4.5 Testování programu

Funkce programu byly otestovány na třech sítích, které jsou popsány v příloze A. U sítí je také zobrazena zpráva o stavovém prostoru a samotný stavový prostor, není-li příliš rozsáhlý. Sítě i vypočítané stavové prostory jsou také uloženy na přiloženém CD ve složce examples.

Jedním z testovaných příkladů je síť popsaná v kapitole 3.4 , zobrazená na obrázku 11. Její kompletní stavový prostor můžeme vidět na obrázku 18.

4.5 Popis podstatných tříd a metod pro ovládání aplikace

4.5.1 Možnost přerušení výpočtu

Stavový prostor některých sítí může být velmi rozsáhlý a jeho výpočet, popřípadě prvotní rozmístění vrcholů jeho grafu, může zabrat nemalou dobu. V případě, že uživatel programu takovýto výpočet začne a pak si jej rozmyslí, má možnost jej přerušit.

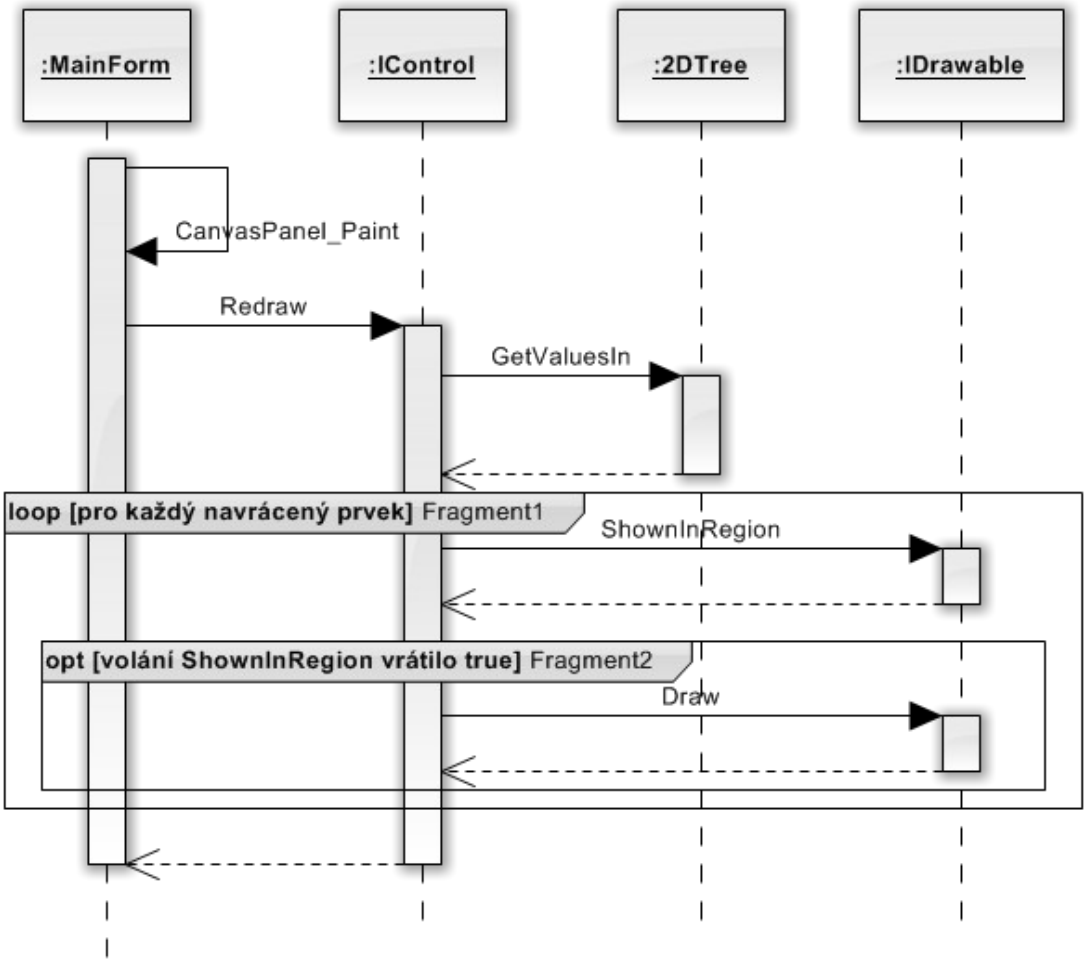
V aplikaci je toto řešeno vytvořením dvou dodatečných vláken. Úkolem prvního vlákna je vytvořit přerušovací dialog reprezentovaný třídou InterruptDialog, úkolem druhého vlákna je provést samotný výpočet. Původní vlákno tyto dvě vlákna spustí a pak čeká, než výpočtové vlákno skončí. Přerušovací dialog se po dobu výpočtu stane jediným prvkem aplikace, který reaguje na uživatelské akce. Při stisku tlačítka přerušovacího dialogu je vyslána zpráva o přerušení čekajícímu hlavnímu vláknu. Při zachycení této zprávy hlavní vlákno přeruší vlákno výpočtové. Pokud zpráva o přerušení nepřijde a výpočtové vlákno svou práci dokončilo, je naopak z hlavního vlákna poslána zpráva o přerušení vláknu přerušovacímu. Tímto se zavře přerušovací dialog a dojde ke zpětné aktivaci hlavního okna aplikace.

4.5.2 Vykreslování

Funkci vykreslování zobrazuje sekvenční diagram na obrázku 19. Tento sekvenční diagram je však z důvodu velikosti a přehlednosti zjednodušený. Hrany grafu jsou uchovávány mimo 2D strom v samostatném seznamu. Jejich vykreslování však probíhá podobně jako u ostatních prvků.

Funkce vykreslování grafu stavového prostoru a jeho pomocných prvků je v aplikaci nejčastěji používaná, a to kdykoliv, když je překresleno okno, když se změní měřítko, když dojde k posunu zobrazení nebo když dojde k posunu či změně viditelnosti prvku grafu. Vzhledem k této skutečnosti jsem se snažil, aby implementace byla co nejméně náročná na čas a systémové prostředky.

Prvním krokem při vykreslování je zjistit, jaká část grafu se překresluje. Při překreslování je metodě OnPaint objektu canvasPanel, na který je vykreslení prováděno, předán parametr ClipBounds, který navrací obdélník reprezentující měněnou oblast. Tato oblast je však vztažena k zobrazovacímu prvku. Pro oblast samotného grafu musíme provést transformaci posunu a měřítka.



Obrázek 19 – Sekvenční diagram vykreslování grafu stavového prostoru

Jakmile je určena oblast grafu, jsou z 2D stromu prvků vybrány ty prvky, které do oblasti potenciálně spadají. Prvky jsou ve 2D stromu klíčovány podle souřadnice levého horního rohu. Pro výběr je tedy nutno připočítat i výšku a šířku největších prvků, aby byly vybrány i prvky, které do oblasti zasahují pouze např. Pravým spodním rohem. Díky tomuto kroku se nemusí procházet všechny prvky grafu a testovat, zda se mají vykreslit. Výjimku tvoří hrany, které jsou pro uchování ve 2D stromu nevhodné a jejich vykreslení nezávisí přímo na tom, zda jsou v oblasti pro vykreslení i její incidentní vrcholy nebo ne. Hrany se vždy prochází všechny, ale volané operace jsou stejné jako pro ostatní prvky.

Po výběru se prvky seřadí podle jejich priority. Priorita zaručuje, že při případném vybírání prvků, které se částečně překrývají, bude vždy vybrán ten, který je opticky navrchu. Pořadí mezi prvky je vždy takové, aby se vykreslily v pořadí hrana grafu → popis hrany → výpis stavového vektoru → vrchol grafu.

Předtím, než dojde k samostatnému vykreslení daného prvku, je ještě proveden test, zda je v oblasti viditelná alespoň část prvku. Může se stát, že prvek je popis hrany, který je na výšku úzký, ale byl vybrán, protože byla původní oblast rozšířená o velikost prvku, který zobrazuje stavový vektor. Tento krok se může zdát zbytečný, jelikož metody pro vykreslování grafických primitiv dokážou prvky ořezávat, ale primitiv pro vykreslení je relativně dost a je zbytečné, aby se prováděly, pokud ve výsledku nemají vliv.

4.5.3 Vybírání prvků

Pro vybírání prvku je použitý podobný princip jako pro vykreslování. Poté, co uživatel klikne myší do plochy grafu, je bod plochy, na který kliknul, převeden do oblasti, ve které se vyskytují prvky, které lze potenciálně vybrat. Pomocí této oblasti se vyberou prvky z 2D stromu pro další zpracování.

Po vybrání se testuje, zda prvek opravdu zasahuje na pozici, na kterou bylo kliknuto, a z těchto prvků se vybere ten, který má nejvyšší prioritu. Priorita je stejná jako při vykreslování, aby bylo dosaženo výběru prvku, který je opticky navrchu.

Pro vybraný prvek, pokud takový je, se pak vyzkouší, zda nebylo kliknuto do oblasti, kde má speciální tlačítko, popřípadě se provede akce tlačítka. Toto je zaručeno speciální metodou IsSpecialClick, kterou definuje rozhraní prvků IMovable. Zároveň se vykoná akce podle stisknutého tlačítka. Vzhledem k tomu, že vybraných prvků může být více, jsou uchovávány v kolekci. Na základě stisknutých kláves Ctrl a Shift a případné přítomnosti v této kolekci se poslední vybraný prvek do této kolekce přidá nebo odebere.

Příhodné je také zmínit, že pokud kliknutím nebyl vybrán žádný prvek, prochází se přes všechny hrany. Pro jejich uchování, jak již bylo zmíněno, není vhodné používat 2D strom. Pro každou hranu se otestuje, jak veliká je její kolmá vzdálenost od bodu kliku. Pokud je nejmenší vypočítaná vzdálenost menší než konkrétní nastavená hranice, přepne se viditelnost popisu odpovídající hrany.

Výběr prvků je možné provádět také blokovým výběrem. V tomto případě jsou prvky vybírány opět jako v případě vykreslování podle oblasti, do které potenciálně spadají. Oblast se vymezí na základě pozice kurzoru při stisku tlačítka a na základě pozice kurzoru při jeho uvolnění. Po výběru se opět testuje jejich reálná přítomnost v oblasti a na základě stisknutých kláves Ctrl a Shift a případné přítomnosti v této kolekci se tyto prvky přidávají nebo odebírají z kolekce všech vybraných prvků.

Tato kolekce je obyčejný seznam na poli, jelikož nepředpokládám, že by počet prvků v této kolekci měl nabývat nevhodně vysoké hodnoty.

4.5.4 Posunování vrcholů a prvků grafu

V aplikaci existují dva druhy posunování. První druh se týká právě označených prvků. Označené prvky jsou uchovávány v seznamu. Pokud uživatel klikne na označený prvek a provede táhnutí, všechny označené prvky se posunou o vektor definovaný body, ve kterých byl kurzor při stisknutí tlačítka, respektive při jeho uvolnění. Při táhnutí se prvky nepohybují plynule, ale přichytávají se ke mřížce. Tohoto je dosaženo odečítáním zbytku po dělení šířkou mřížky od délky posunu, a to pro každou osu zvlášť.

Druhý druh posunování se týká všech prvků v daném rozsahu souřadnic. Uživatel vybere souřadnici na dané ose, která v dalším kroku slouží jako hranice, a provede táhnutí. Všechny prvky, jež mají odpovídající souřadnici větší než hranice, se posunou o vzdálenost, která odpovídá průmětu vektoru definovaného body, ve kterých byl kurzor při stisknutí tlačítka, respektive při jeho uvolnění, na odpovídající osu.

V obou druzích posunování je potřeba aktualizovat umístění prvků ve 2D stromu. Toto je prováděno tak, že se prvek před posunem vyhledá, odebere ze stromu, změní se jeho souřadnice a nově se do stromu vloží.

Při posunování vrcholu grafů se zároveň nově vypočítají parametry úseček, které reprezentují incidentní hrany grafu. Tímto se omezí potřeba přepočítávat je při každém vykreslování.

4.5.5 Ukládání a nahrávání vypočítaného stavového prostoru

V aplikaci je možné vypočítaný stavový prostor ukládat a otevírat nezávisle na síti, pro kterou byl vypočítán.

Pro ukládání jsem se rozmýšlel mezi ukládáním v binární podobě nebo ve formátu XML. Rozhodl jsem se pro binární podobu, jelikož se mi zdá jednodušší na sestavení i čtení, zvláště za pomoci tříd BinaryWriter a BinaryReader, jež poskytuje platforma .NET. Nevýhody tohoto řešení jsou samozřejmě problémy při změně formátu a nekompatibilita souboru s jinými programy. Jelikož však informace v grafu stavového prostoru považuji za dostatečné, změny formátu nepředpokládám.

Následuje podrobný popis posloupnosti údajů, které jsou do souboru vkládány. Řetězce jsou zapsány v kódování UTF-8, přičemž jejich délka je zapsána před prvním znakem jako čtyřbajtová little-endian hodnota (int32), celočíselné hodnoty jsou ukládány jako čtyřbajtová little-endian hodnota (int32), čísla s plovoucí řádovou čárkou jsou ukládány jako čtyřbajtová little-endian hodnota (float, nebo také single), logická hodnota je ukládána jako jednobajtová hodnota (0 nebo 1), kolekce jsou ukládány jako opakující se posloupnost svých prvků, přičemž prvnímu prvku předchází čtyřbajtová little-endian hodnota (int32) udávající celkový počet prvků.

První záznam je kontrolní řetězec „StateSpace_~~\n“, který určuje, zda se opravdu jedná o platný soubor. Následují informace o stavovém prostoru v tomto pořadí:

1) text zprávy o stavovém prostoru,
2) šířka nejširšího prvku grafu,
3) výška nejvyššího prvku grafu,
4) kolekce vrcholů grafu,
5) kolekce hran grafu.

Pro každý vrchol grafu v kolekci jsou uchovávány tyto informace v pořadí:

1) kolekce míst a pro každé místo kolekce značek, které na něm v tomto stavovém vektoru jsou
2) příznak, zda jsou všechny značky na výstupním místě,
3) číslo stavu, který tento vrchol reprezentuje,
4) X-ová a Y-ová pozice vrcholu
5) viditelnost vrcholu,
6) X-ová a Y-ová pozice ukotvení prvku, ve kterém se vypisuje stavový vektor, který odpovídá danému stavu
7) X-ová a Y-ová pozice výše uvedeného prvku,
8) viditelnost výše uvedeného prvku.

Pro každou hranu grafu v kolekci jsou uchovávány tyto informace v pořadí:

1) X-ová a Y-ová pozice zdrojového vrcholu grafu,
2) X-ová a Y-ová pozice cílového vrcholu grafu,
3) název provedeného přechodu,
4) údaj o chování instancí formuláře zpráv,
5) kolekce názvů proměnných a barev značek, které přenáší,
6) X-ová a Y-ová pozice ukotvení prvku, ve kterém se vypisují informace o hraně,
7) X-ová a Y-ová pozice výše zmíněného prvku,
8) viditelnost výše zmíněného prvku.

Aby se zamezila degenerace 2D stromu, ve kterém jsou uchovávány prvky, je při načítání použit stejný algoritmus jako při vkládání prvků nově vypočítaného stavového prostoru. Algoritmus popisuje obrázek 16.

4.5.6 Zpět a opakovat

Při operacích zpět a opakovat je použito podobného principu jako pro zápis do souboru, tedy použití tříd BinaryWriter a BinaryReader pro zápis do streamu. V tomto případě se ale jedná o stream paměťový místo souborového a objem zapisovaných dat je

výrazně omezen. Změny v grafu stavového prostoru zahrnují pouze polohu a viditelnost prvků, jejich počet se nemění. Stačí tedy uchovávat pouze hodnoty, které se týkají polohy a viditelnosti. Není ani potřeba ukládat reference na objekty, ke kterým údaje patří, jelikož v pomocných seznamech od jejich nahrání po jejich smazání jsou tyto objekty stále umístěny na stejné pozici.

Jednotlivé streamy s reprezentací grafu stavového prostoru jsou uchovávány v lineárním seznamu a je alokován ukazatel na položku, která reprezentuje aktuální vzhled. Pokud je provedena změna, zruší se všechny streamy v seznamu za aktuální položkou, aktuální vzhled se zapíše na následující položku a ukazatel se na tuto položku posune.

Rozsah historie je vždy omezen a proto se vždy testuje, zda je možno provést krok zpět nebo dopředu podle pozice ukazatele. Tento test je prováděn vždy, když by se měl vyvolávat stav z historie. V aplikaci je využito vlastností CanUndo a CanRedo pro umožnění nebo naopak znemožnění používání tlačítek s touto funkčností.

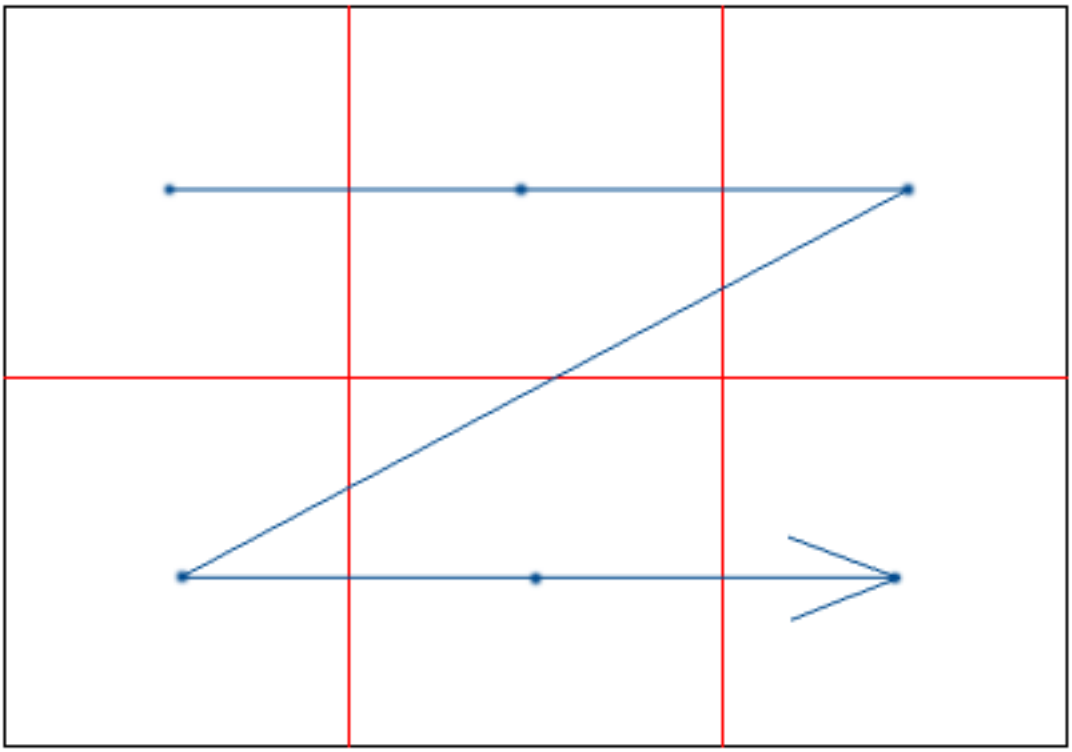
4.5.7 Export do obrázku

Funkce pro export do obrázku funguje velmi podobně jako funkce vykreslení. V tomto případě se ale vykreslují všechny viditelné prvky a to v měřítku 1:1. Před samotným vykreslováním je potřeba připravit kreslící plochu, která je v tomto případě obrázek. Pro tento účel se hledají prvky s minimálními a maximálními souřadnicemi. Obrázek se vytvoří takový, aby se na něj vešly všechny viditelné prvky a přidá se okraj o velikost 10 pixelů ke každé hraně. Výsledný obrázek je pak uložen ve formátu PNG.

4.5.8 Tisk

Funkce tisku zároveň využívá funkci exportu do obrázku. Při tisku je totiž potřeba znát rozměry grafu, aby šlo zjistit, zda je potřeba tisknout další stránku, a zároveň oblast tisku. Na obrázek tedy jednou vykreslíme všechny prvky grafu. Při tisku se pak pouze vyjme požadovaná oblast obrázku a není třeba vyhledávat a testovat viditelnost každého prvku zvlášť.

Při tisku se z obrázku vynechávají vygenerované 10px okraje. V metodě pro tisk se pracuje s palci jako základní jednotkou, protože na tuto jednotku lze rozměry převádět jak z parametru PageSettings, tak z rozměrů obrázku. Graf je tištěn po sloupcích a pak po řádcích. Názorně je schéma tisku vykresleno na příkladu na obrázku 20, kde černý obdélník představuje hranice grafu, červená mřížka představuje hranice tiskových stránek a modrá šipka představuje pořadí, v jakém jsou stránky tištěny.



Obrázek 20 – Schéma tisku

Jelikož platforma .NET neposkytuje komplexní dialog pro nastavení tisku, je použit dialog pouze jednoduchý, ve kterém lze vybrat tiskárnu, orientaci stránky, popřípadě další parametry podle tiskárny (např. rozlišení či velikost okrajů).

5 Možnosti propojení s editorem ABA-CPN

Aplikace editoru sítě ABA-CPN je druhá část praktické části, kterou tvoří Bc. Petr Nejman. Propojení těchto aplikací nebyl kritický požadavek. Z pohledu uživatelského komfortu je však možnost kontroly stavového prostoru pro právě editovanou sít velmi výhodná.

Po vzájemných konzultacích s Bc. Nejmanem a vedoucím práce jsme se shodli, že nepůjdeme cestou úplné integrace jakou poskytuje nástroj CPN Tools, tedy nebudeme zobrazovat graf stavového prostoru na stejnou plochu, na které je graf sítě. Rozhodnutí padlo na základě těchto úvah:

- Je rychlejší a přehlednější, když jsou oba grafy oddělené v samostatných oknech a existuje možnost ponechat zobrazení v okně na jednom místě, na které se návrhář sítě soustředí.
- Obě aplikace byly do značné míry vyvíjeny nezávisle na sobě a fungují jako samostatné aplikace. Proto se mohou objevit jisté nekompatibility se zobrazováním prvků grafu stavového prostoru na ploše editoru.

Samotné propojení aplikací jsme provedli tak, že jsme zdrojové kódy aplikace pro výpočet stavového prostoru vložili do aplikace editoru jako samostatný modul. Hlavní aplikací celého programu je tedy editor a z něj je možné vyvolávat výpočet stavového prostoru. Naopak to možné není.

5.1 Přístupové metody

Protože se na aplikaci pro výpočet stavového prostoru pohlíží jako na samostatný modul, je přístup prováděn pouze přes ovládací třídu MainForm jmenného prostoru CPNStateTool. MainForm obsahuje několik veřejných prvků a metod, ke kterým je možné z editoru přistupovat. Tyto jsou následující:

- metoda LoadNet,
- metoda LoadOnly.
- metoda Refresh,
- delegát StreamGenerator typu StreamCallerDelegate.

Pomocí metody LoadNet předá editor mé části aplikace v nějakém formátu síť, se kterou se právě pracuje, a aplikace ihned spočítá její stavový prostor. Metoda má několik přetížení. První přetížení má paramtery ICPN, IkolekceMnozinBarev, IkolekcePromennych a IkolekceKonstant. Tato metoda je pro propojení základní. Touto metodou se do aplikace předá síť přesně v tom stavu, v jakém se nachází v editoru. Na předanou síť se odkazuje přímo. To znamená, že v obou aplikacích se pracuje se stejnými objekty. Jelikož výpočet stavového prostoru neběží vzhledem k editoru v samostatném vlákně, není potřeba synchronizace.

Druhé přetížení má parametr Stream. Tato slouží jako záložní metoda, kdyby se objevily nečekané a složité komplikace s řešením, kdy se předají přímé odkazy na objekty. Předává se pouze XML stream, jež obsahuje předpis sítě. Tento stream lze použít pro rekonstrukci sítě podobně jako soubor.

Poslední přetížení má parametry object a EventArgs. Tato slouží pouze jako EventHandler pro nabídku načtení sítě. Ve verzi aplikace integrované s editorem není tato metoda využívaná vůbec.

Metoda LoadOnly slouží pro nahrání sítě do aplikace pro výpočet stavového prostoru, aniž by se ihned počítal její stavový prostor. Při volání metody LoadNet je pro nahranou síť ihned vypočítán stavový prostor. V některých případech toto chování není vhodné. Například v editoru je historie změn implementována tak, že se vytváří nové instance objektů, které představují síť. Aby objekty zůstávaly aktuální, je potřeba je stále předávat. V případě komplikované sítě by výpočet stavového prostoru mohl trvat neúměrně dlouhou dobu a pokud by počet vracených kroků měl být větší najednou, tyto výpočty by ani nebyly potřeba.

Metoda Refresh slouží pro přepočítání stavového prostoru. Aplikace si pamatuje, z jakého zdroje přišla poslední síť a podle toho se v této metodě zachová. Byla-li síť předána přímo, tak pouze vypočítá nový graf stavového prostoru. Pokud však byla síť předána přes stream, nepředal se přímý odkaz, a proto je potřeba před každým výpočtem zažádat o aktuální stream z editoru a generovat novou síť. Pro tento účel existuje delegát StreamGenerator, do kterého editor uloží svou metodu, pomocí které vygeneruje stream z aktuální sítě.

5.2 Společná rozhraní a struktury

Po dohodě s Bc. Petrem Nejmanem jsme zavedli společná rozhraní, která využíváme pro přístup k síti CPN. Tyto jsou

- ICPN, které zastřešuje přístup k síti CPN,
- IMisto, IPrechod a IHrana, které představují místa, přechody, respektive hrany sítě CPN,
- IVrchol, IEementObrazu, IVlastnikTextu, které doplňují rozhraní IMisto, IPrechod a IHrana,
- IMnozinaBarev a IKolekceMnozinBarev, které představují množiny barev sítě CPN a jejich kolekci,
- IPromenna a IKolekcePromennych, které představují proměnné sítě CPN a jejich kolekci,
- IKonstanta a IKolekceKonstant, které představují konstanty sítě CPN a jejich kolekci,
- DruhyHrany, DruhPrechodu a DruhMista, výčtové typy, jež určují, jakého typu je daná hrana, daný přechod respektive dané místo.

V aplikaci výpočtu stavového prostoru jsou implementující třídy omezeny pouze na ty vlastnosti a metody rozhraním definované, které jsou přímo potřeba, nebo které lze implementovat implicitně (tzn. jednoduché gettery a settery). Při přístupu k jiným vlastnostem či vyvolání jiné metody je vyvolána výjimka NotImplementedException, tudíž tyto třídy nelze použít v editoru. Naopak to možné je, jelikož implementující třídy předávané editorem implementují rozhraní kompletně. Typickým příkladem takových vlastností a metod jsou vlastnosti navracející třídy VnitrniBod a Voditko nebo metoda Vykresli.

5.3 Modul verifikace

Jeden z návrhů funkcí při propojení aplikací byl verifikace sítě před začátkem výpočtu přímo v aplikaci pro výpočet stavového prostoru. Návrh řešení spočíval v použití delegátu, kterým by se z editoru předal odkaz na funkci, která verifikaci provádí. Při tomto řešení se ale občas vracela hláška, že v síť nemá přípustné inicializační značení. Pro výpočet stavového prostoru je to chyba nepodstatná, jelikož při výpočtu se existující inicializační značení nebere v potaz. Proto se od tohoto návrhu upustilo a domluvili jsme se, že si modul, který slouží pro verifikaci, mohu zkopírovat a upravit podle vlastních potřeb. Tímto řešením je zaručeno, že síť, pro kterou se počítá stavový prostor, je verifikována, i když by aplikace pro výpočet stavového prostoru běžela samostatně.

Více jsem o této problematice psal v kapitole 4.4.2 .

6 Závěr

Splnil jsem zadání práce. Vytvořil a otestoval jsem aplikaci, která dokáže vypočítat kompletní stavový prostor podtřídy barvených Petriho sítí ABA-CPN. Tuto aplikaci jsme s Bc. Petrem Nejmanem úspěšně propojili s jeho editorem sítí ABA-CPN. Tímto jsme vytvořili ucelený softwarový nástroj, který umožňuje specializovanou práci na této podtřídě Petriho sítí.

Popsal jsem, co je architektura simulačních modelů ABAsim v kapitole 2 , včetně jejích základních prvků – agentů. Dále jsem shrnul problematiku Petriho sítí v kapitole 3 , od klasických P/T Petriho sítí přes barvené Petriho sítě až po podtřídu ABA-CPN, která je navržena pro popis chování agentů v architektuře ABAsim.

Dále jsem podrobně popsal aplikaci, a sice její hlavní funkci – výpočet stavového prostoru v kapitole 4.4.3 , další funkce ve ostatních podkapitolách kapitoly 4 a propojení s editorem sítí v kapitole 5 .

7 Použitá literatura

[1] KAVIČKA, A.; KLIMA, V.; ADAMNKO, A. *Agentovo orientovaná simulácia dopravných uzlov.* Žilina : EDIS, 2005. 206 s. ISBN 80-8070-477-5.

[2] ČEŠKA, Milan. *Petriho sítě*. Brno : CERM, 1994. 94 s. ISBN 80-85867-35-4.

[3] JENSEN, K.: *Coloured Petri Nets. Basic Concepts, Analysis Method and Practical Use. Volume 1. (Barvené Petriho sítě. Základní pojmy, analytické metody a praktické použití. První svazek.)*, druhé vydání, 1997, ISBN: 3-540-60943-1

[4] KAVIČKA, A.; ŽARNAY, M.. Specifikace a analýza podtřídy barvené Petriho sítě pro aplikace v rámci simulačních modelů dopravních systémů. *Perner´s Contacts*, 2008, vol. 3, no. 5, s. 153-162. ISSN: 1801-674X.

[5] VESELÝ, P.. *Interpret barvené petriho sítě v rámci ABAsim architektury simulačních modelů*. [s.l.], 2007. 53s s. Diplomová práce. Univerzita Pardubice. Vedoucí práce Ing. Jiří Pinkas.

[6] KAVIČKA, A.. *Agentové orientované architektury simulačních modelů*. [s.l.], 2010. 43 s. Syllabus přednášek. Univerzita Pardubice.

[7] KAVIČKA, A.. *Petriho sítě : Formalismus vhodný k popisu dynamiky systémů*. [s.l.], 2010. 35 s. Syllabus přednášek. Univerzita Pardubice.

[8] KAVIČKA, A.; ŽARNAY, M. Aplication of coloured Petri net for agent control and communication in the ABASIM architecture. In *In proceeding of the 9th workshop and tutorial on practical use of coloured Petri nets and the CPN tools*. Aarhus : University of Aarhus (Denmark), 2008. s. 47-62. ISSN 0105-8571.

[9] *CPN Tools Homepage* [online]. 2010 [cit. 2011-04-28]. Dostupné z WWW: <http://cpntools.org/>.

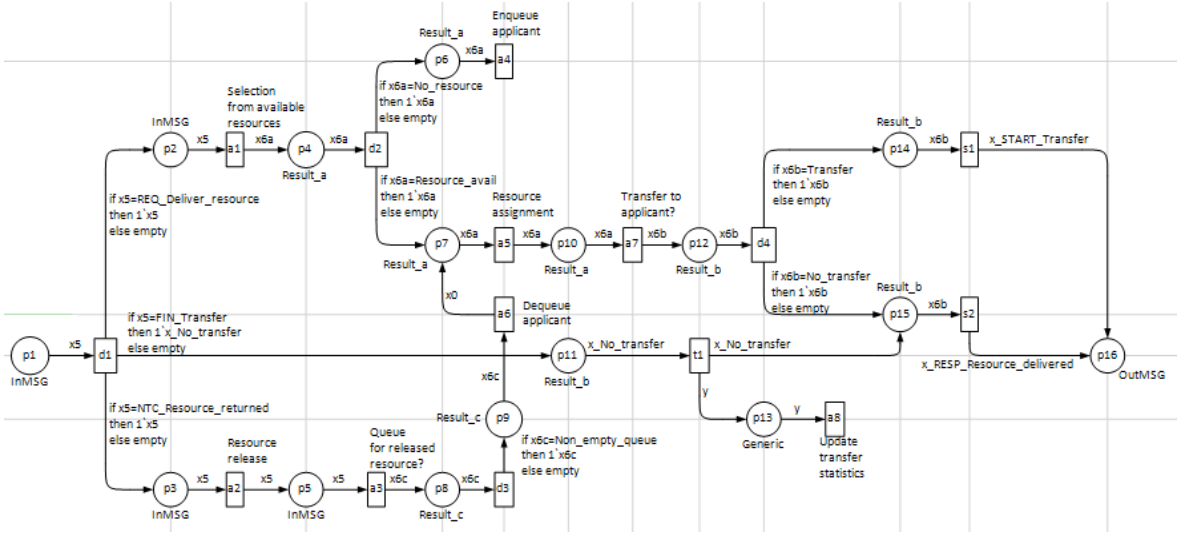
[10] *MSDN Library : .NET Framework Class Library* [online]. c2011 [cit. 2011-05-05]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/gg145045.aspx>.

Přílohy

Obsah příloh

Příloha A: Testované sítě

Síť 1: Jednoduchá síť



Obrázek 21 – Jednoduchá síť

Pro síť jsou definovány následující množiny barev:

- *InMSG={REQ_Deliver_resource, NTC_Resource_returned, FIN_Transfer}*,
- *OutMSG={RESP_Resource_delivered, START_Transfer}*,
- *Result_a={Resoure_avail, NoResource}*,
- *Result_b={Transfer, No_transfer}*,
- *Result_c={Empty_queue, Non_empty_queue}*,
- *Generic={e}*,

následující konstanty:

- *x0=Resource_avail*,
- *x_RESP_Resource_delivered=RESP_Resource_delivered*,
- *x_START_Transfer=START_Transfer*,
- *x_No_transfer=No_transfer*,

a následující proměnné:

- *y* : *Generic*,
- *x5* : *InMSG*,
- *x6a* : *Result_a*,

- *x6b* : *Result_b*,
- *x6c* : *Result_c*.

Zpráva o stavovém prostoru

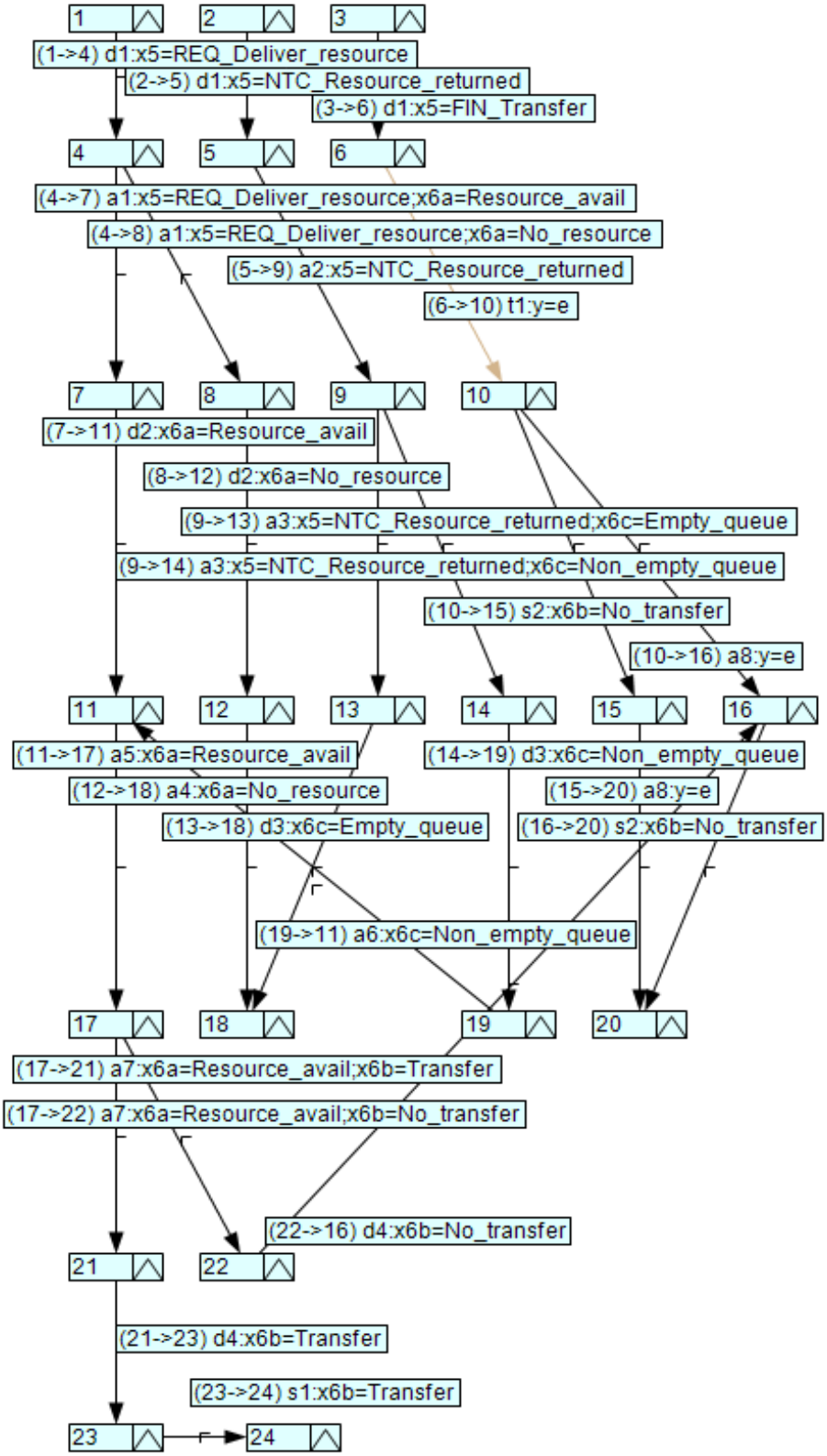
Statistika:
Počet stavových vektorů: 24
Počet stavových změn: 25
Počet vstupních stavů: 3
Počet koncových stavů: 3
- z toho nekorektních: 0

Rozmezí hodnot:

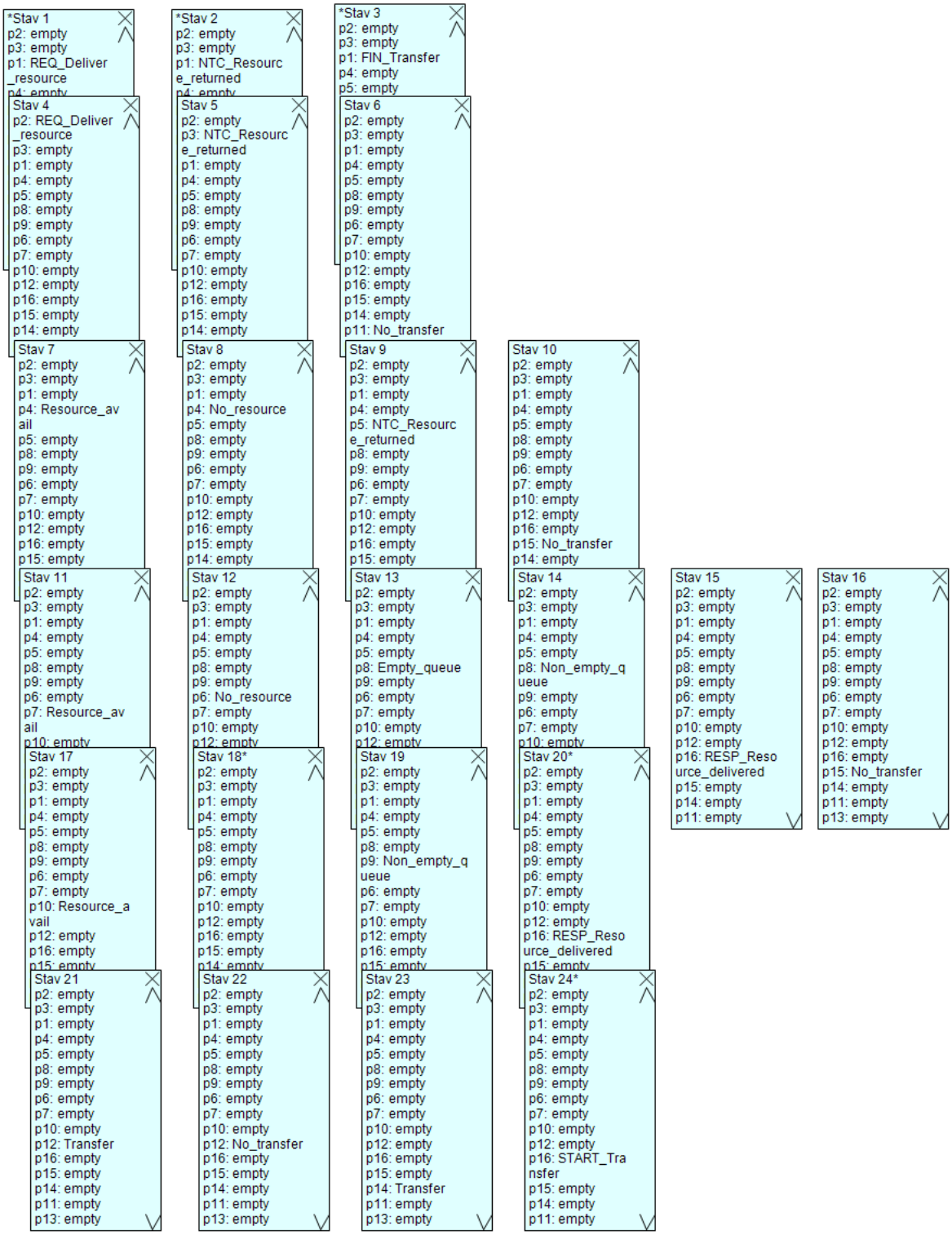
Místo	Maximum	Hodnoty
p2	1	REQ_Deliver_resource (ve stavu 4);
p3	1	NTC_Resource_returned (ve stavu 5);
p1	1	REQ_Deliver_resource (ve stavu 1);
		NTC_Resource_returned (ve stavu 2);
		FIN_Transfer (ve stavu 3);
p4	1	Resource_avail (ve stavu 7);
		No_resource (ve stavu 8);
p5	1	NTC_Resource_returned (ve stavu 9);
p8	1	Empty_queue (ve stavu 13);
		Non_empty_queue (ve stavu 14);
p9	1	Non_empty_queue (ve stavu 19);
p6	1	No_resource (ve stavu 12);
p7	1	Resource_avail (ve stavu 11);
p10	1	Resource_avail (ve stavu 17);
p12	1	Transfer (ve stavu 21);
		No_transfer (ve stavu 22);
p16	1	RESP_Resource_delivered (ve stavu 15);
		RESP_Resource_delivered (ve stavu 20);
		START_Transfer (ve stavu 24);
p15	1	No_transfer (ve stavu 10);
		No_transfer (ve stavu 16);
p14	1	Transfer (ve stavu 23);
p11	1	No_transfer (ve stavu 6);
p13	1	e (ve stavu 10);
		e (ve stavu 15);

Informace o živosti:
Koncové stavy (nekorektní jsou označeny podtržítky): [18, 20, 24]
Neprovedené přechody: (žádné)

Stavový prostor



Obrázek 22 – Stavový prostor jednodušší sítě

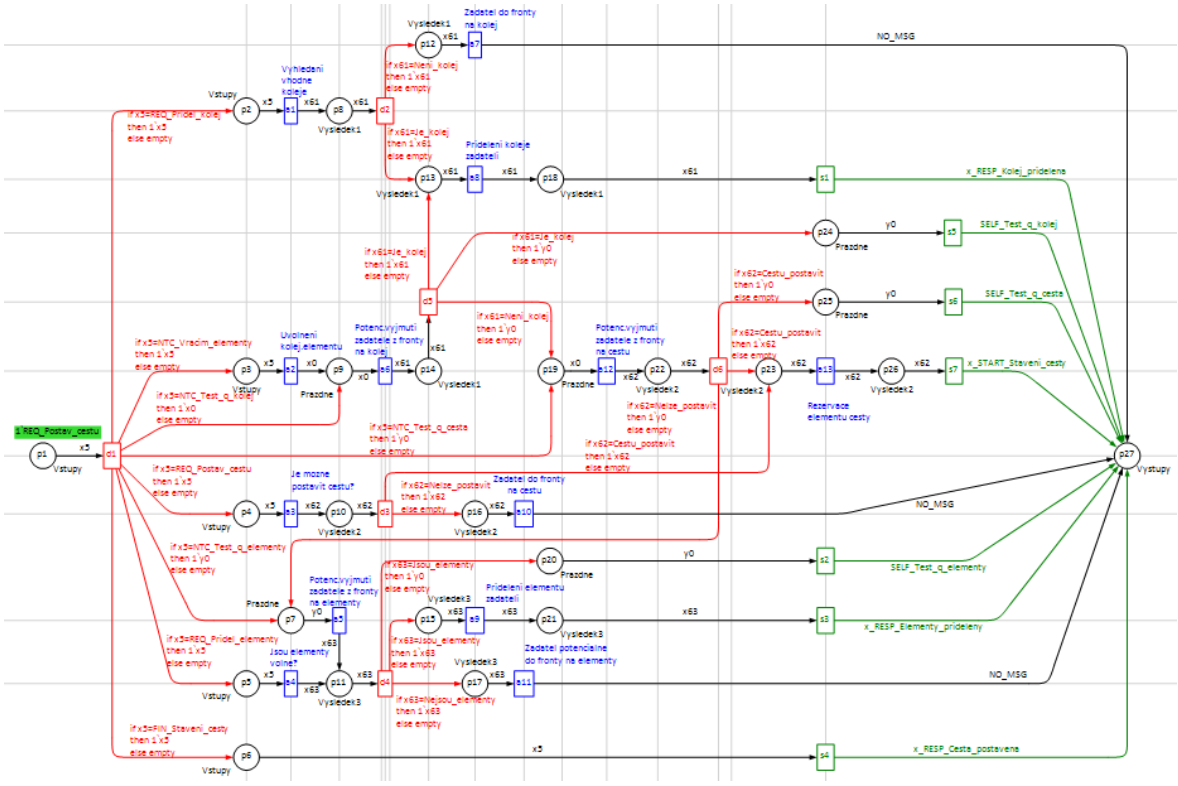


Obrázek 23 – Stavové vektory stavového prostoru jednodušší sítě

Síť 2: Rozsáhlejší síť

Pro síť jsou definovány následující množiny barev:

- *Vstupy={REQ_Pridel_kolej, NTC_Vracim_elementy, REQ_Postav_cestu, FIN_Staveni_cesty, REQ_Pridel_elementy, NTC_Test_q_elementy, NTC_Test_q_cesta, NTC_Test_q_kolej}*,
- *Vysledek1={Je_kolej, Neni_kolej}*,
- *Vysledek2={Cestu_postavit, Nelze_postavit}*,
- *Vysledek3={Jsou_elementy, Nejsou_elementy}*,
- *Vystupy={RESP_Cesta_postavena, RESP_Kolej_pridelena, START_Staveni_cesty, RESP_Elementy_prideleny, NO_MSG, SELF_Test_q_elementy, SELF_Test_q_cesta, SELF_Test_q_kolej}*,
- *Prazdne={e}*,



Obrázek 24 – Rozsáhlejší síť

následující konstanty:

- *x_RESP_Cesta_postavena= RESP_Cesta_postavena,*
- *x_RESP_Kolej_pridelena=RESP_Kolej_pridelena*,

- *x_RESP_Elementy_prideleny=RESP_Elementy_prideleny*,
- *x_START_Staveni_cesty=START_Staveni_cesty,*

a následující proměnné:

- *x5* : *Vstupy*,
- *x0* : *Prazdne*,
- *y0* : *Prazdne*,
- *vystup* : *Vystupy*,
- *x61* : *Vysledek1,*
- *x62 : Vysledek2,*
- *x63 : Vysledek3*.

Zpráva o stavovém prostoru

Statistika:
Počet stavových vektorů: 55
Počet stavových změn: 60
Počet vstupních stavů: 8
Počet koncových stavů: 7
- z toho nekorektních: 0

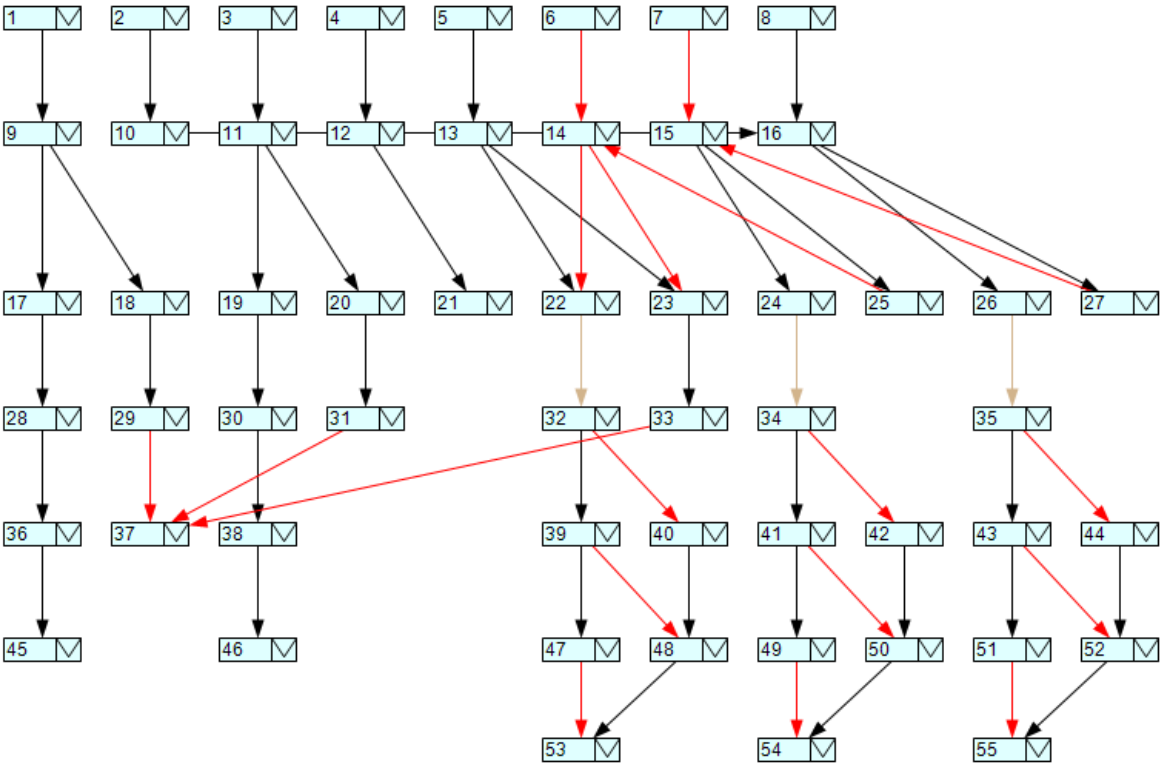
Rozmezí hodnot:

Místo	Maximum	Hodnoty
p2	1	REQ_Pridel_kolej (ve stavu 9);
p4	1	REQ_Postav_cestu (ve stavu 11);
p1	1	REQ_Pridel_kolej (ve stavu 1);
		NTC_Vracim_elementy (ve stavu 2);
		REQ_Postav_cestu (ve stavu 3);
		FIN_Staveni_cesty (ve stavu 4);
		REQ_Pridel_elementy (ve stavu 5);
		NTC_Test_q_elementy (ve stavu 6);
		NTC_Test_q_cesta (ve stavu 7);
		NTC_Test_q_kolej (ve stavu 8);
p8	1	Je_kolej (ve stavu 17);
		Neni_kolej (ve stavu 18);
p10	1	Cestu_postavit (ve stavu 19);
		Nelze_postavit (ve stavu 20);
p12	1	Neni_kolej (ve stavu 29);
p13	1	Je_kolej (ve stavu 28);
		Je_kolej (ve stavu 35);
		Je_kolej (ve stavu 44);

p27	2	RESP_Elementy_prideleny, SELF_Test_q_elementy (ve stavu 53);
		SELF_Test_q_cesta, START_Staveni_cesty (ve stavu 54);
		RESP_Kolej_pridelena, SELF_Test_q_kolej (ve stavu 55);
p3	1	NTC_Vracim_elementy (ve stavu 10);
p9	1	e (ve stavu 16);
p19	1	e (ve stavu 15);
p22	1	Cestu_postavit (ve stavu 24);
		Nelze_postavit (ve stavu 25);
p23	1	Cestu_postavit (ve stavu 30);
		Cestu_postavit (ve stavu 34);
		Cestu_postavit (ve stavu 42);
p16	1	Nelze_postavit (ve stavu 31);
p5	1	REQ_Pridel_elementy (ve stavu 13);
p11	1	Jsou_elementy (ve stavu 22);
		Nejsou_elementy (ve stavu 23);
p17	1	Nejsou_elementy (ve stavu 33);
p15	1	Jsou_elementy (ve stavu 32);
		Jsou_elementy (ve stavu 40);
p7	1	e (ve stavu 14);
p14	1	Je_kolej (ve stavu 26);
		Neni_kolej (ve stavu 27);
p18	1	Je_kolej (ve stavu 36);
		Je_kolej (ve stavu 43);
		Je_kolej (ve stavu 52);
p21	1	Jsou_elementy (ve stavu 39);
		Jsou_elementy (ve stavu 48);
p6	1	FIN_Staveni_cesty (ve stavu 12);
p26	1	Cestu_postavit (ve stavu 38);
		Cestu_postavit (ve stavu 41);
		Cestu_postavit (ve stavu 50);
p20	1	e (ve stavu 32);
		e (ve stavu 39);
		e (ve stavu 47);
p25	1	e (ve stavu 34);
		e (ve stavu 41);
		e (ve stavu 49);
p24	1	e (ve stavu 35);
		e (ve stavu 43);
		e (ve stavu 51);

Informace o živosti:
Koncové stavy (nekorektní jsou označeny podtržítky): [21, 37, 45, 46, 53, 54, 55]
Neprovedené přechody: (žádné)

Stavový prostor



Obrázek 25 – Stavový prostor rozsáhlejší sítě

Z důvodu rozsáhlosti stavového prostoru není uveden jeho podrobnější popis.

Síť 3: Síť s mnoha možnostmi větvení stavového prostoru

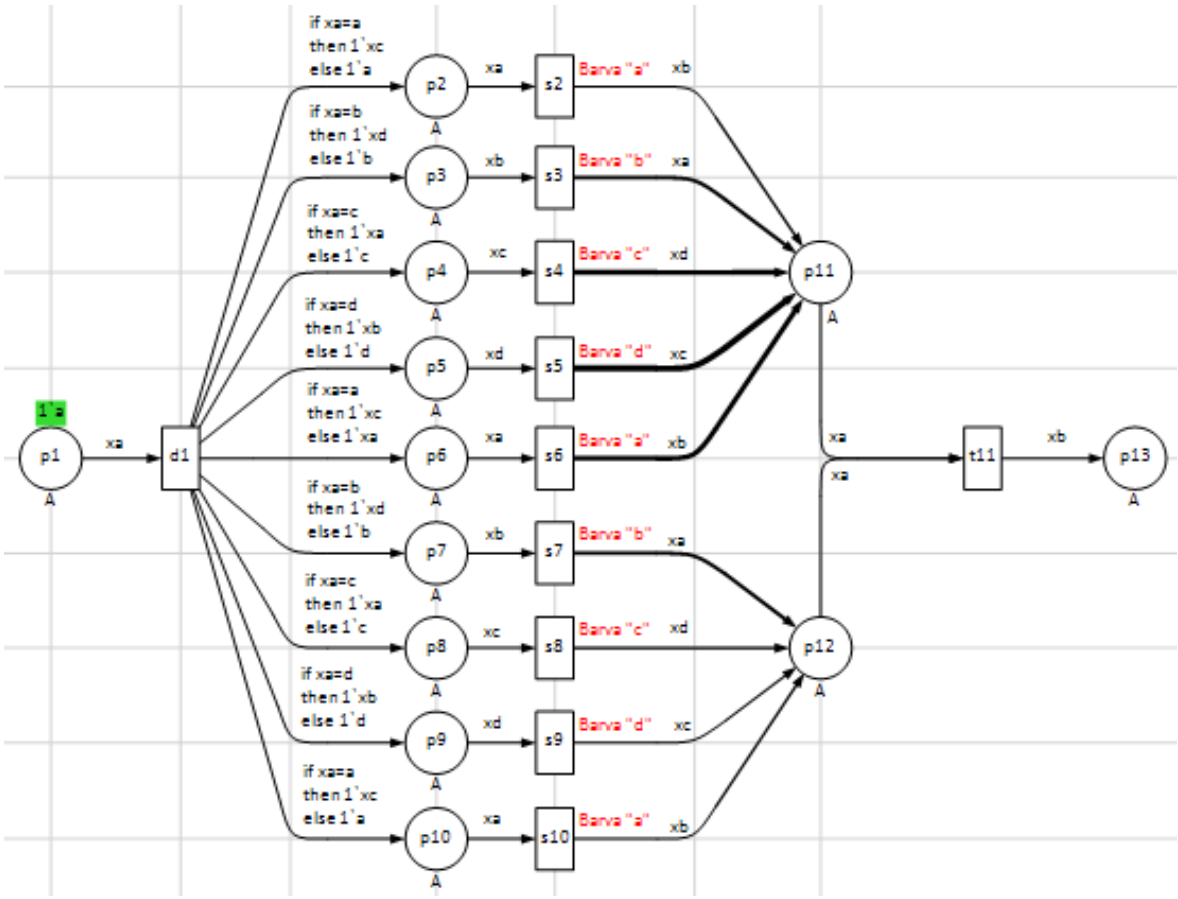
Na přiloženém CD je síť uložena pod názvem vetveni.cpn

Pro síť je definována následující množina barev:

- $A=\{a, b, c, d, e, f, g, h, i\}$,

a následující proměnné:

- $xa : A$,
- $xb : A$,
- $xc : A$,
- $xd : A$.



Obrázek 26 – Síť s mnoha možnostmi větvení stavového prostoru

Zpráva o stavovém prostoru

Statistika:
Počet stavových vektorů: 11759
Počet stavových změn: 53634
Počet vstupních stavů: 9
Počet koncových stavů: 9
- z toho nekorektních: 9

Informace o rozmezích hodnot byly vypuštěny z důvodu nadměrného rozsahu.

Informace o živosti:
Koncové stavy (nekorektní jsou označeny podtržítky): [_11751_, _11752_, _11753_, _11754_, _11755_, _11756_, _11757_, _11758_, _11759_]
Neprovedené přechody: (žádné)

Nekorektní koncové stavy jsou dány tím, že v místě *p11* vždy zbude jedna značka, ke které se neobjeví odpovídající značka v místě *p12*.

Stavový prostor

Stavový prostor sítě je příliš rozsáhlý a proto zde není uveden.

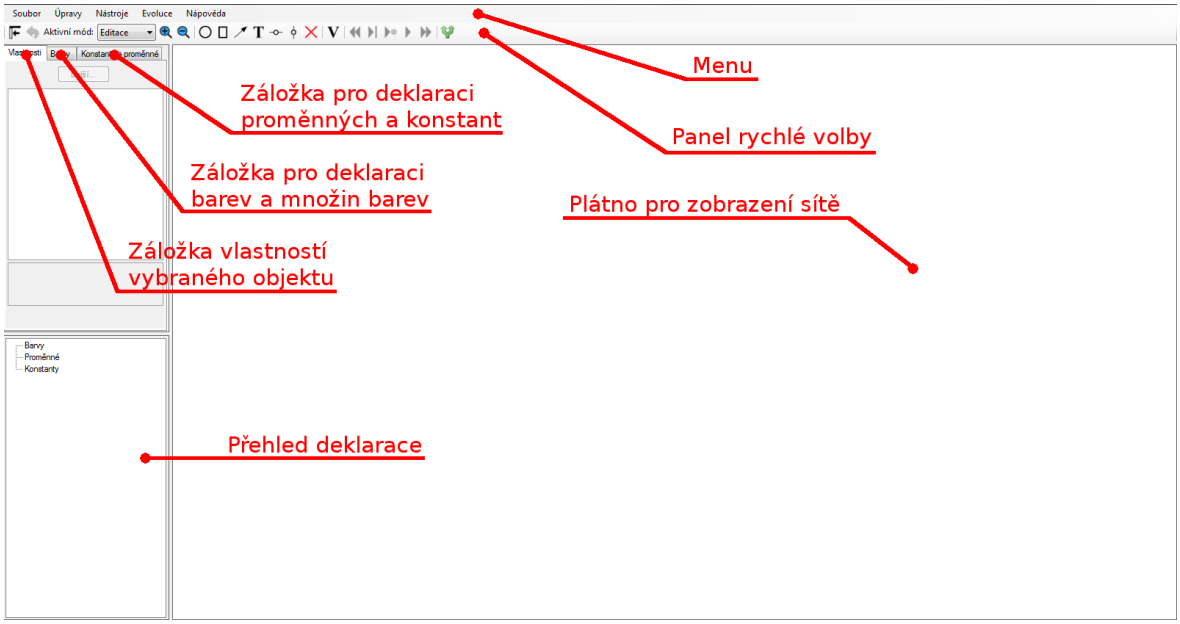
Příloha B: Uživatelská příručka k programu

Nápověda k aplikaci je dodávána s programem v podobě HTML stránek. Každá stránka odpovídá jedné z hlavních kapitol této přílohy. V aplikaci pro výpočet stavového prostoru lze HTML nápovědu vyvolat z nabídky Nápověda → Obsah nebo tlačítkem nápovědy na panelu tlačítek rychlého spouštění.

Text nápovědy v této příloze je shodný s nápovědou dodávanou s aplikací. V první části je nápověda pro editor, ve druhé pak nápověda aplikace pro výpočet stavového prostotu.

Uživatelské rozhraní editoru

Rozložení jednotlivých panelů je vidět na obrázku 27.



Obrázek 27 – Rozložení aplikace editoru

Dále jsou detailně popsány jednotlivé panely a jejich možnosti.

Menu

V menu je možno vybrat si z možností *Soubor, Úpravy, Nástroje, Evoluce* a *Nápověda*.

Možnost *Soubor* dává k dispozici následující akce:

- *Nový* – Smaže veškeré informace o předešlé zobrazené síti a vytvoří novou prázdnou síť. Klávesová zkratka *ctrl+N*.
- *Otevřít…* – Otevře dialogové okno, pro výběr souboru s příponou *cpn*, který se po potvrzení výběru souboru pokusí otevřít a zobrazit síť. Klávesová zkratka *ctrl+O*.

- *Uložit* – Pokud už otevřená síť byla uložena, uloží ji znovu na stejné místo. Pokud síť zatím nebyla uložena, pracuje stejně jako možnost *Uložit jako…*. Klávesová zkratka *ctrl+S*.
- *Uložit jako…* – Otevře dialogové okno pro výběr lokace a názvu souboru, do kterého se po potvrzení dialogu síť uloží. Klávesová zkratka *ctrl+L*.
- *Ukončit* – Ukončí celou aplikaci. Klávesová zkratka *alt+F4*.

Možnost *Úpravy* dává k dispozicí následující akce:

- *Zpět* – Vrátí poslední změnu provedenou nad sítí (nevztahuje se na evoluci sítě). Klávesová zkratka *ctrl+Z*.
- *Přiblížit* – Přiblíží pohled na síť. Klávesová zkratka *ctrl+Dolu*.
- *Oddálit* – Oddálí pohled na síť. Klávesová zkratka *ctrl+Nahoru*.
- Možnosti Přiblížit a oddálit se také dají vyvolat pomocí otáčení kolečka myši.

Možnost *Nástroje* dává k dispozicí následující akce:

- *Vlož místo* – Po aktivaci se po přejetí myši nad *Plátno* objeví na pozici kurzoru kruh, vykreslený přerušovanou čarou. Tento kruh se pohybuje stejně jako kurzor myši a je tak možné vybrat pozici vkládaného místa. Po nalezení vhodného místa, stiskněte pro vložení místa levé tlačítko myši. Klávesová zkratka *ctrl+M*.
- *Vlož přechod* – Je to obdobná akce jako *Vlož místo*, jen se místo kružnice objevuje na pozici kurzoru obdélník. Klávesová zkratka *ctrl+P.*
- *Vlož hranu* – Po aktivaci vytvoříte hranu tak, že nad výchozím vrcholem zamýšlené hrany stisknete a držíte levé tlačítko myši. Potom za stálého držení tohoto tlačítka přesunete kurzor myši nad cílový vrchol, kde držené tlačítko pustíte. Během držení tlačítka by se měla objevit přerušovaná čára, vycházející z výchozího vrcholu, která ústí do místa kurzoru myši. Klávesová zkratka *ctrl+H.*
- *Vlož textovou poznámku* – Je to obdobná akce jako *Vlož přechod*. Text poznámky se potom upraví pomocí záložky vlastností. Klávesová zkratka *ctrl+T.*
- *Vlož horizontální vodítko* – Je to obdobná akce jako *Vlož přechod*, jen se místo obdélníku objevuje horizontální přerušovaná čára přes celé *Plátno*. Klávesová zkratka *ctrl+I.*
- *Vlož vertikální vodítko* – Je to obdobná akce jako *Vlož přechod*, jen se místo obdélníku objevuje vertikální přerušovaná čára přes celé *Plátno*. Klávesová zkratka *ctrl+J.*

- *Odeber element* – Po aktivaci stačí levým tlačítkem myši kliknout na libovolný element na *Plátně* a ten se odebere. Všechny elementy nelze odebrat přímo např. popisek hrany lze odebrat pouze tak, že se odebere celá hrana. Klávesová zkratka *ctrl+D*.
- *Verifikace* – Provede verifikaci sítě, zobrazí okno s výpisem chyb a chyby červeně vyznačí i přímo na vykreslené síti. Klávesová zkratka *ctrl+V*.
- Možnost *Evoluce* dává k dispozicí následující akce:
- *Zpět na začátek* – Přesune stav evoluce zpět na inicializační značení. Klávesová zkratka *ctrl+B*.
- *Automatické provedení přechodu* – Po aktivaci je možno kliknutím levého tlačítka myši na proveditelný přechod provést náhodně právě tento přechod. Kliknutím levého tlačítka myši kamkoli jinam způsobí náhodné vybrání proveditelného přechodu a jeho provedení. Klávesová zkratka *ctrl+A*.
- *Kontrolované provedení přechodu* – Po aktivaci je možno kliknutím levého tlačítka myši na proveditelný přechod provést právě tento přechod. Následně se otevře dialogové okno, jehož možnosti jsou vysvětleny dále s možností nastavit jednotlivé proměnné a konstanty účastnící se provádění přechodu. Kliknutím levého tlačítka myši kamkoli jinam na *Plátno* se libovolný přechod nevybere. Klávesová zkratka *ctrl+K*.
- *Spustit evoluci* – Po aktivaci této možnosti se začnou jednotlivé proveditelné přechody postupně náhodně vybírat a provádět. Postupně se tak provede jakási prezentace evoluce aktuální sítě. Klávesová zkratka *ctrl+X*.
- *Rychlé provedení sítě* – Je to obdobná akce jako *Spustit evoluci.* Jen mezi prováděním jednotlivých přechodů není žádná časová prodleva a tak se evoluce sítě provede „okamžitě“. Klávesová zkratka *ctrl+R*.

Možnost *Nápověda* dává k dispozicí následující akce:

- *Obsah* – Zobrazí obsah nápovědy pomocí webového prohlížeče. Klávesová zkratka *F1*.
- *O aplikaci* – Zobrazí základní informace o aplikaci (součást nápovědy) pomocí webového prohlížeče. Klávesová zkratka *ctrl+Q*.

Panel rychlé volby

V panelu rychlé volby jsou k dispozici následující akce:

- Tlačítko pro schování panelu – Schová nebo zobrazí celý panel vlevo od *Plátna*.
- Tlačítko zpět – Stejná akce jako *Úpravy → Zpět*.

- Přepínač režimu – Přepíná mezi režimy editoru. Do režimu *Evoluce* se lze úspěšně přepnout jen tehdy, když síť projde bez nalezených chyb verifikací. Režimy editoru budou vysvětleny níže.
- Přiblížení – Stejná akce jako *Úpravy → Přiblížit*.
- Oddálení – Stejná akce jako *Úpravy → Oddálit*.
- Místo – Stejná akce jako *Nástroje → Vlož místo*.
- Přechod – Stejná akce jako *Nástroje → Vlož přechod*.
- Hrana – Stejná akce jako *Nástroje → Vlož hranu*.
- Poznámka – Stejná akce jako *Nástroje → Vlož textovou poznámku*.
- Horizontální vodítko – Stejná akce jako *Nástroje → Vlož horizontální vodítko*.
- Vertikální vodítko – Stejná akce jako *Nástroje → Vlož vertikální vodítko*.
- Odebrání elementu – Stejná akce jako *Nástroje → Odeber element*.
- Verifikace – Stejná akce jako *Nástroje → Verifikace*.
- Vrátit na začátek – Stejná akce jako *Evoluce → Zpět na začátek*.
- Automatické provedení přechodu – Stejná akce jako *Evoluce → Automatické provedení přechodu*.
- Kontrolované provedení přechodu – Stejná akce jako *Evoluce → Kontrolované provedení přechodu*.
- Spuštění evoluce – Stejná akce jako *Evoluce → Spustit evoluci*.
- Rychlé provedení přechodu – Stejná akce jako *Evoluce → Rychlé provedení přechodu*.
- Stavový prostor – Otevře ABA-CPN State Space a vypočítá stavový prostor aktuální sítě.

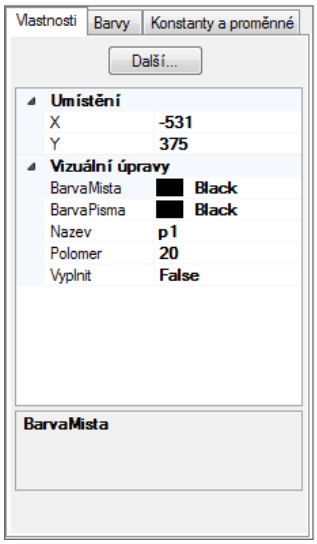
Plátno pro zobrazení sítě

Práce s tímto plátnem byla již částečně probrána v přehledu možných akcí (např. vkládání místa). Nebylo však zmíněno jak pohybovat s již vloženými elementy a sítí jako celkem. S celou sítí resp. pohledem na síť lze pohnout, pokud najedete kurzorem myši kamkoli na plátno mimo jakýkoli vložený element. Potom za držení pravého tlačítka myši lze pohybovat s celým pohledem na síť. Hýbat s již vloženými elementy lze až na jednu výjimku pouze v režimu *Editace*. Stačí kurzorem myši najet nad vybraný element, se kterým potřebujeme pohnout a podobně jako s celou sítí za držení pravého tlačítka myši posunout vybraný element na vybrané místo. S hranami se takto ale hýbat nedá a tak místo posunu celé hrany se na místě chycení hrana ohýbá. Pokud je hrana chycena za již vytvořený ohyb, hýbeme právě s tímto ohybem a nový se nevytváří. Jak již bylo zmíněno,

tak můžeme se vším hýbat až na jednu výjimku pouze v režimu *Editace*. Touto výjimkou jsou značky (v zeleném poli) umístěné na libovolném místě. V průběhu evoluce by se mohly ukázat na nevhodné pozici, kde by zakrývali důležitou část sítě a je tedy nutné aby s nimi bylo možné pohybovat i v průběhu evoluce. Při pokusu o posunutí libovolného jiného elementu v režimu *Evoluce* se posunuje celá síť. Poslední funkcionalitou plátna je označení libovolného elementu. Během tohoto označení nesmí být aktivní žádný nástroj (zvlášť pozor na nástroj Odeber element). Levým tlačítkem myši klikněte na libovolný element, tento se podbarví modrou barvou. Pokud je aktivní záložka vlastností, tak na ní můžete vidět a většinou i měnit libovolnou vlastnost, ale o této záložce později. Toto označování funguje pouze v režimu *Editace*.

Záložka vlastností vybraného objektu

Tato záložka ukazuje vlastnosti vybraného elementu. Na obrázku jsou jako příklad zobrazeny vlastnosti místa. Většina vlastností lze pomocí této záložky změnit, ale některé jsou zde jen pro čtení a svojí hodnotu nastavují pomocí dialogu. Takovým příkladem je například popis rozhodovací hrany, kde vlastnost s názvem *Obsah* ukazuje text vypsaný na plátně u hrany. Tento text je vygenerovaný pomocí dialogu, do kterého se lze dostat, pokud označíme příslušnou hranu a stiskneme tlačítko *Další…*, které je také součástí záložky *Vlastnosti*. O těchto dialogových oknech později. Může se také stát, že při přepisování některé přepisovatelné vlastnosti se po potvrzení objeví opět původní text. Je to z toho důvodu, že zadaný text nebyl přijat kvůli filtru, který zjišťuje nepovolené znaky, jako je např. diakritika. Této aplikaci by tyto znaky problém nedělaly, ale jelikož pracuje s xml soubory, které jsou zdrojem i pro CPN Tools, který potom tento soubor není schopen otevřít, tak je tu tento filtr zaveden.

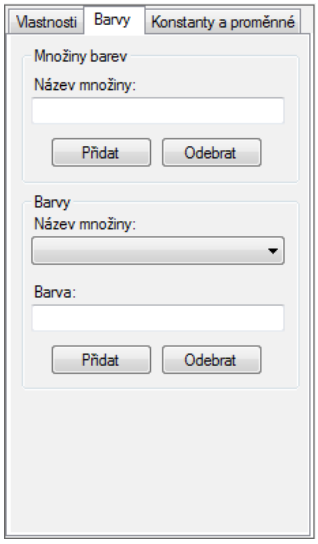


Obrázek 28 – Záložka vlastnosti

Záložka pro deklaraci barev a množin barev

Tato záložka je určena pro deklaraci barev a množin barev. Je logicky rozdělena do dvou sekcí.

První sekce s názvem *Množiny barev* slouží pro manipulaci s celými množinami barev. Pro vložení nové množiny stačí do textového pole s označením *Název množiny* zapsat název vkládané množiny barev a následně stisknout tlačítko *Přidat*. Pro odebrání množiny barev se musí její název objevit ve stejném textovém poli jako při vkládání. Toho docílíme buď ručně, nebo stačí kliknout levým tlačítkem myši na tento název v přehledu deklarace popsaném níže a název se nám objeví na potřebném místě. Potom už jen stačí stisknout tlačítko *Odebrat* v příslušné sekci a množina je odebrána.



Obrázek 29 – Záložka deklarace barev

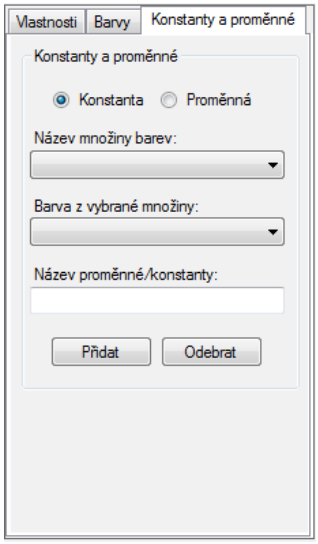
Druhá sekce s názvem *Barvy* slouží pro manipulaci přímo s jednotlivými barvami. Vkládání a odebírání jednotlivých barev probíhá podobně jako vkládání a odebírání celých množin. Navíc se zde musí uvést pro kterou množinu barev tuto změnu (např. vložení barvy) děláme. Množinu si jednoduše nalezneme v této sekci v ComboBoxu s označením *Název množiny*. Při odebírání si název barvy můžeme opět jednoduše vyplnit kliknutím na název barvy v přehledu deklarace.

Při libovolném odebírání jak množiny barev, tak i barvy se může objevit informativní okno o odebrání některé závislé proměnné nebo konstanty. Při odebrání např. množiny barev jsou proměnné, které mají definici podle této množiny následně definovány podle neexistující množiny barev a tak se odeberou. Podobně jsou na tom i konstanty, o jejichž odebrání vám zmíněné informativní okno také podá informaci.

Záložka pro deklaraci proměnných a konstant

Tato záložka slouží pro definici konstant a proměnných. Při vkládání se nejprve nastaví pomocí horního přepínače, zda se vloží konstanta nebo proměnná. Potom se pomocí ComboBoxu označeného jako *Název množiny barev*, vybere přidělená množina barev. Následně jen v případě vkládání konstanty vybereme pomocí druhého ComboBoxu konkrétní barvu, která bude přidělena konstantě. V případě tvorby proměnné je tento ComboBox nefunkční (není potřeba). Nakonec jen stačí vyplnit ve spodním textovém poli název vytvářené proměnné či konstanty a stisknout tlačítko *Přidat*. Pro odebrání slouží

podobně jako v předchozí záložce vyplněný název v textovém poli, který lze opět získat stejným způsobem jako v předcházejících případech z přehledu deklarace.



Obrázek 30 – Záložka deklarace proměnných a konstant

Přehled deklarace

V tomto panelu je znázorněna veškerá deklarace v podobě stromové struktury. První kořen stromu *Barvy* obsahuje na další úrovni názvy jednotlivých deklarovaných množin barev. Další úroveň pod množinami obsahuje už konkrétní barvy z dané množiny barev. Druhý kořen stromu *Proměnné* obsahuje na další úrovni deklarované proměnné v podobě *název : množina_barev*, kde na pozici *název* leží název proměnné a za dvojtečkou místo *množina_barev* leží název množiny barev, ze které je daná proměnná. Třetí strom s kořenem *Konstanty* obsahuje pochopitelně na další úrovni deklarované konstanty. Konstanty jsou zde zapisovány ve tvaru *název = barva*, kde *název* podobně jako v předchozím případě znázorňuje název konstanty a *barva* přiřazenou barvu, kterou tato konstanta obsahuje.

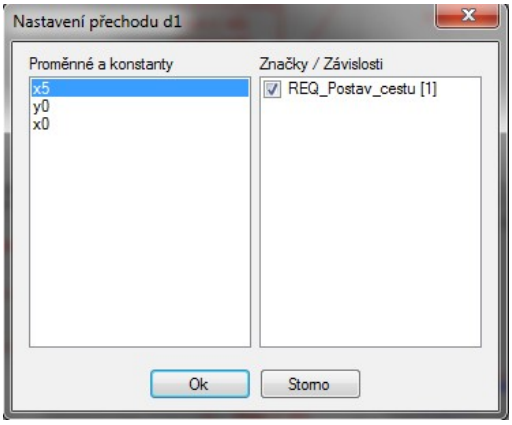
Režimy editoru

V průběhu této nápovědy bylo o režimech editoru řečeno snad už vše, takže jen pro shrnutí. Existují zde dva režimy *Editace* a *Evoluce*, mezi kterými lze přepínat pomocí ComboBoxu umístěném v panelu rychlé volby. Do režimu *Evoluce* se lze přepnout jen, pokud je síť bez chyby. Režim *Editace* sloužím pro vytváření a úpravy sítě. Během tohoto režimu nejsou přístupné akce související s evolucí sítě. Režim *Evoluce* je určen k provádění evoluce vytvořené sítě. Během tohoto režimu nejsou k dispozici nástroje pro úpravu a vkládání nových elementů do sítě, nelze označovat elementy a ani nelze s většinou elementů hýbat.

Dialogová okna

Dialogové okno pro nastavení provedení přechodu

Toto dialogové okno se otevírá, při použití nástroje *Kontrolované provedení přechodu* pokud je tento nástroj použit na některý proveditelný přechod. V levé části okna jsou jednotlivé proměnné a konstanty, které se účastní provádění přechodu, pro který byl tento dialog vytvořen. V pravé části okna jsou možnosti, právě označené proměnné nebo konstanty z levé části. Po výběru jedné z možností u všech proměnných a konstant stačí stisknout tlačítko *Ok* a přechod se podle nastavení provede. Tlačítkem *Storno* se z dialogu vyskočí bez provádění přechodu.



Obrázek 31 – Dialog provedení přechodu

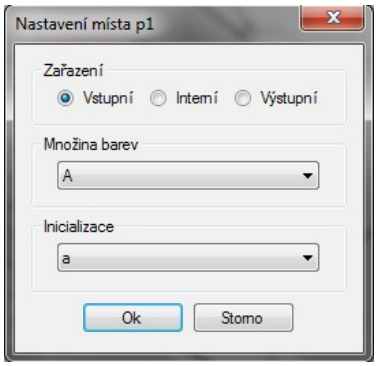
Zápis proměnných nebo konstant v levé části je následující. Je zde zapsán pouze název nebo v případě nutného odlišení stejné proměnné nebo konstanty je za názvem v kulatých závorkách zapsán i název incidentního místa, se kterým je konkrétní proměnná či konstanta spojena.

Zápis možností v pravé části okna je následující. Může zde být zapsán jen název proměnné nebo konstanty z levé části, ze které se přebírá instance i značka. Další možností je zápis ve tvaru *barva [instance]*, kde *barva* znázorňuje název barvy a *instance* identifikuje číslo instance. Instance může ale být zapsána trojím způsobem. První nejjednodušší způsob je pouze číslem, kdy je vidět přesně o jakou instanci jde. Dalším způsobem je, že na místě instance je název proměnné nebo konstanty. To značí, že instance bude od této konstanty či proměnné přebrána. Poslední co se může na místě instance objevit je slovo *nová*, což značí, že se zde vytvoří pro danou barvu nová instance.

Dialogové okno pro nastavení místa

Toto dialogové okno se otevře při označení libovolného místa a následně při stisknutí tlačítka *Další…* v záložce vlastností. Lze zde nastavit o jaké místo se jedná (vstupní, interní, výstupní) pomocí přepínače v horní části okna. Zde můžeme nastavit množinu barev, pro kterou je místo definováno, pomocí ComboBoxu v sekci *Množina barev*. Poslední věc, která tímto dialogem lze nastavit, je inicializační značka. Ta se nastaví pomocí druhého ComboBoxu a to jen pokud je vybrána v horní části *Zařazení* možnost

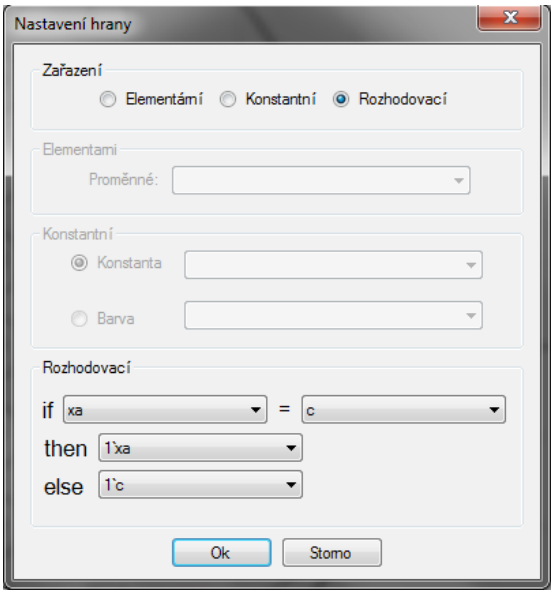
Vstupní. To značí, že je místo vstupním místem a je tedy nutné vybrat mu inicializační značení. Nakonec tlačítkem *Ok* potvrdíme změny, nebo tlačítkem *Storno* opustíme dialogové okno beze změn.



Obrázek 32 – Dialog nastavení místa

Dialogové okno pro nastavení hrany

Tento dialog slouží k bližšímu nastavení hrany. Opět se zde rozděluje dialog na několik sekcí, kde je aktivní vždy jen ta potřebná. Potřebná sekce je ta, která souvisí se zařazením hrany (Elementární, Konstantní, Rozhodovací). Toto zařazení lze měnit v horní části dialogového okna pomocí tří přepínačů.



Obrázek 33 – Dialog nastavení hrany

První sekce se věnuje elementárním hranám, které na sebe váží proměnné. Tudíž se zde může vybrat pomocí ComboBoxu proměnná, která se dané hraně přidělí. Proměnné v tomto ComboBoxu procházejí filtrem, který zde dovoluje jen ty proměnné, které mohou vstupovat nebo vystupovat z incidentního místa. Pokud incidentní místo dosud nemá definovanou množinu barev, jsou zde k dispozici všechny proměnné.

Druhá sekce se zabývá konstantními hranami. Tato hrana na sebe váže pouze konstanty. Tyto konstanty mohou být buď přímo deklarované konstanty, nebo jednoduše

deklarované barvy. Mezi těmito dvěma možnostmi lze opět vybírat pomocí přepínačů, umístěných v této sekci. Konstanty i barvy lze vybírat pomocí příslušných ComboBoxů, ve kterých je stejně jako v předchozí sekci filtrován obsah.

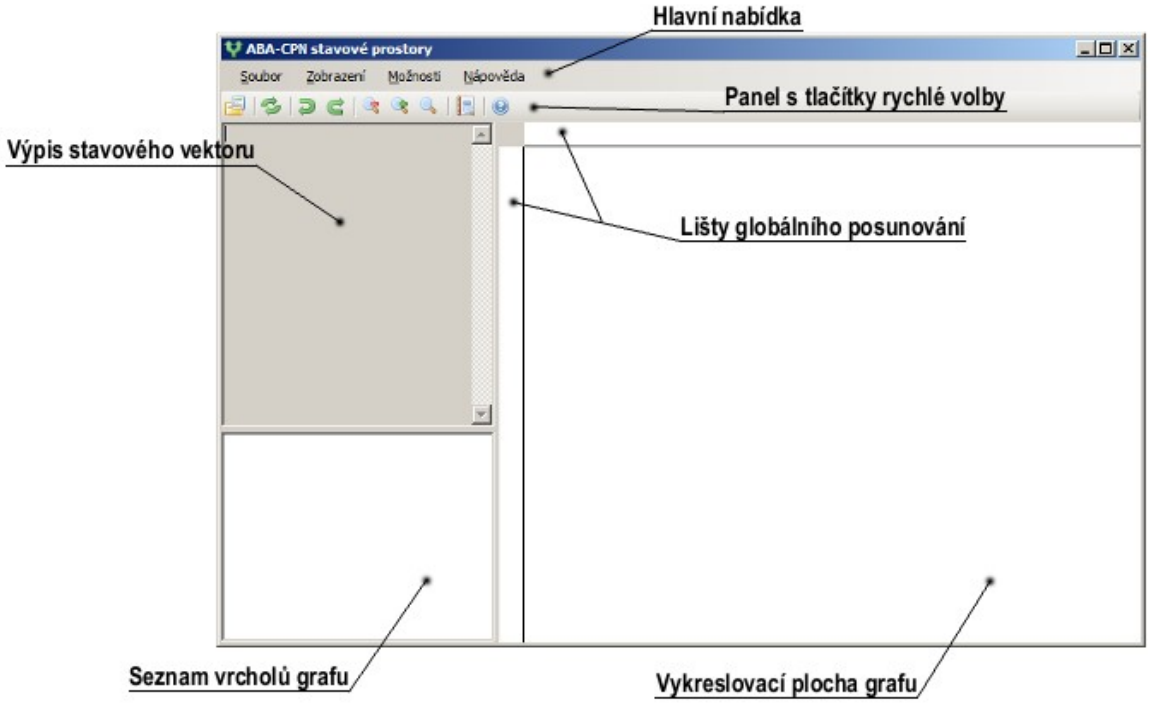
Poslední sekce nastavuje rozhodovací hranu. Je postavena pomocí ComboBoxů, které jsou uspořádány tak, že je vidět, kterou část rozhodovacího výrazu vyplňují. ComboBox umístěný za *if* obsahuje jen proměnné, umístěné na vstupu incidentního rozhodovacího přechodu. ComboBox umístěný za znakem = obsahuje jen barvy z množiny barev vybrané proměnné z předchozího ComboBoxu. ComboBoxy umístěné za *then* a *else* mají stejnou náplň a to všechny barvy, proměnné a konstanty, které mohou vstoupit do incidentního místa. Mimo zmíněných je obsahem i klíčové slovo *empty*, které značí žádný přenos. Na tuto náplň ComboBoxů je zde znovu obdobný filtr jako v předchozích sekcích.

Po nastavení příslušné sekce stačí potvrdit změny pomocí tlačítka *Ok*, nebo opustit dialog beze změn pomocí tlačítka *Storno*.

Další část nápovědy se týká aplikace pro výpočet stavového prostoru.

Rozhraní aplikace pro výpočet stavového prostoru

Rozhraní aplikace je zobrazeno na obrázku 34.



Obrázek 34 – Rozhraní aplikace

Hlavní nabídka obsahuje všechny možné operace, které lze vykonat (Viz Soubor, Zobrazení a Možnosti).

Na panelu s tlačítky rychlé volby jsou umístěny ty operace, které předpokládám, že budou často využívány (Viz Akční tlačítka).

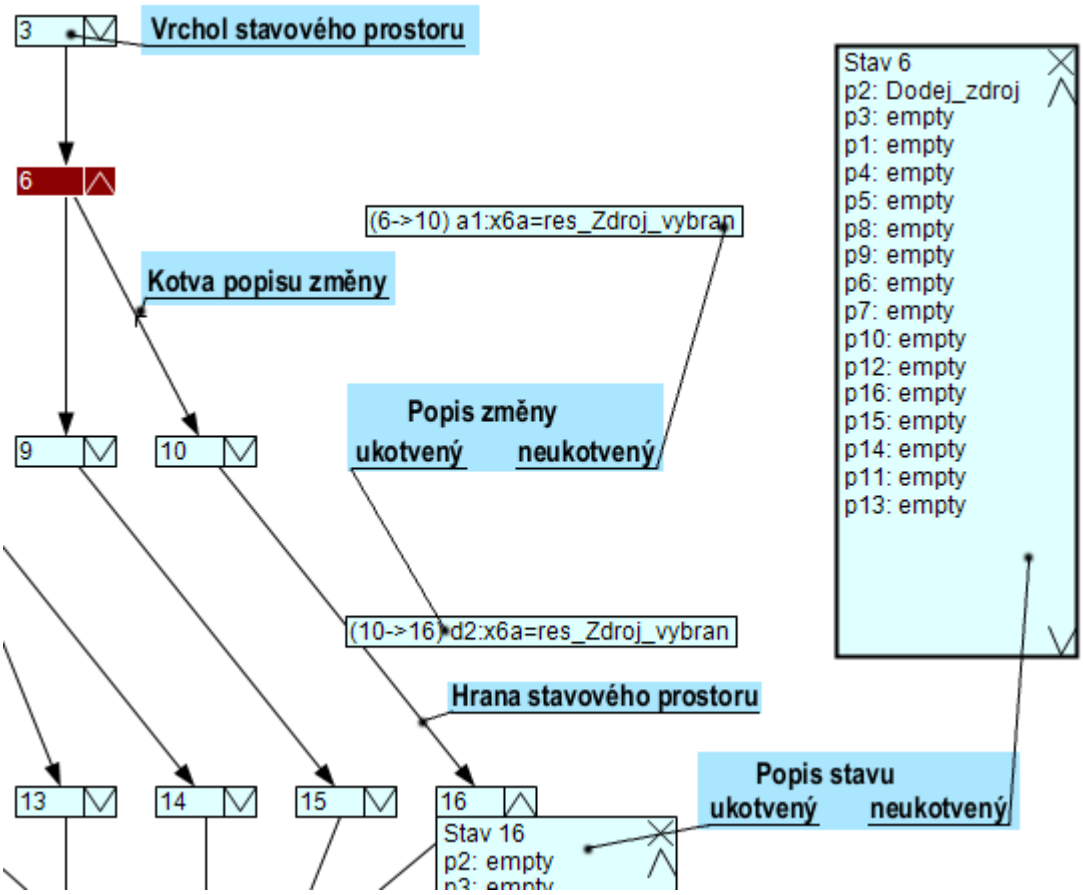
Na vykreslovací plochu grafu se vykresluje graf stavového prostoru.

Výpis stavového vektoru obsahuje seznam míst a značek reprezentovaný posledním vybraným vrcholem grafu.

Seznam vrcholů grafu obsahuje seznam všech vrcholů a umožňuje je vybírat, vyhledat nebo upravit jejich viditelnost.

Lišty globálního posunování slouží k současnému posunu více prvků grafu (Viz Grafická úprava grafu).

Prvky grafu jsou zobrazeny na obrázku 35.



Obrázek 35 – Prvky grafu

Vrchol grafu představuje vypočítaný stav. Tlačítko na pravé straně prvku zobrazuje a schovává prvek, který slouží k výpisu odpovídajícího stavu.

Hrana grafu představuje změnu stavu. Klikem na ni se zobrazuje a schovává prvek, který slouží k výpisu dodatečných informací o změně.

Výpis stavu zobrazuje dodatečné informace o stavu. Tlačítko "×" prvek schovává, tlačítko "∧" posouvá řádky výpisu o řádek výše a tlačítko "∨" posouvá řádky výpisu o řádek níže. Tento prvek je ukotven, pokud je jeho levý okraj na stejné horizontální souřadnici jako levý okraj odpovídajícího vrcholu a zároveň pokud je jeho horní okraj na stejné vertikální souřadnici jako spodní okraj odpovídajícího vrcholu.

Popis změny zobrazuje informace o změně stavu. Tyto jsou číslo zdrojového a cílového stavu, provedený přechod a hodnoty proměnných souvisejících se změnou. Tento prvek je ukotven, pokud je jeho levý horní roh umístěn na odpovídající kotvě popisu změny. Kotva se zobrazuje pouze pokud je popis změny viditelný

Ukotvení znamená, že při posunu vrcholu nebo hrany se zároveň posune odpovídající informační prvek.

Pokud je prvek barevně odlišný, je označený. Pro každý označený prvek je zvýrazněn obrys odpovídajícího souvisejícího prvku. Související prvky jsou vrchol <–> výpis stavu a hrana <–> popis změny.

Barvy a velikosti prvků lze měnit, viz Nastavení.

Nabídka Soubor

Nabídka soubor obsahuje hlavně operace pro práci s okolím aplikace.

Nabízené operace jsou následující:

- Otevřít uložený prostor

  Zobrazí otevírací dialog, ve kterém vyberete soubor .stsp. z vybraného souboru se pak aplikace pokusí rekonstruovat předem vypočítaný graf stavového prostoru. Zároveň provede obnovení zobrazení.

- Uložit prostor

  Zobrazí ukládací dialog, ve kterém vyberete soubor .stsp. Do vybraného souboru se pak aplikace pokusí zapsat aktuálně vypočítaný graf stavového prostoru.

- Přepočítat síť

  Přinutí aplikaci, aby přepočítala stavový prostor sítě. Zároveň provede obnovení zobrazení. Je-li síť načtena ze souboru, před výpočtem se načte znovu.

- Vytisknout

  Zobrazí dialog nastavení tiskárny. Na základě nastavení se pak pokusí vytisknout aktuálně vypočítaný graf stavového prostoru.

- Konec

  Ukončí aplikaci.

Nabídka Zobrazení

Nabídka zobrazení obsahuje hlavně operace související s prezentací grafu.

Nabízené operace jsou následující:

- Zpět

Navrátí poslední změnu grafu stavového prostoru – polohu prvků nebo jejich viditelnost.

- Opakovat

  Opakuje poslední změnu grafu stavového prostoru – polohu prvků nebo jejich viditelnost.

- Přiblížit

  Přiblíží graf (zvětšení).

- Oddálit

  Oddálí graf (zmenšení).

- Obnovit

  Obnoví přiblížení na původní (1:1) a nastaví souřadnice levého hroního rohu na [0,0].

- Export do PNG

  Otevře dialog Pro ukládání, ve kterém vyberete soubor .png. do vybraného souboru se pak aplikace pokusí zapsat obrázek typu PNG, který zobrazuje aktuálně vypočítaný graf stavového prostoru.

Nabídka Možnosti

Nabídka soubor obsahuje ostatní operace, které je možné vykonávat.

Nabízené operace jsou následující:

- Zobrazit Report

  Zobrazí dialog, ve kterém se vypíše text zprávy o aktuálně vypočítaném stavovém prostoru sítě. Z tohoto dialogu je možné zprávu uložit.

- Uložit report

  Zobrazí ukládací dialog, ve kterém vyberete soubor .txt. Do vybraného souboru se pak aplikace pokusí zapsat text zprávy o aktuálně vypočítaném stavovém prostoru sítě.

- Předvolby

  Zobrazí dialog, který obsahuje možnosti pro nastavení velikostí a barev prvků grafu stavového prostoru.

Funkční tlačítka

Tato tlačítka představují operace, které předpokládám, že budou nejčastěji využívány. Jejich ikony odpovídají ikonám v nabídce. Zleva doprava jsou to operace:

- přepočítat stavový prostor právě otevřené sítě,

- vrátit poslední úpravu grafu,
- opakovat poslední úpravu grafu,
- oddálit graf,
- přiblížit graf,
- obnovit původní zobrazení,
- zobrazit zprávu o stavovém prostoru,
- zobrazit tuto nápovědu.

Popis těchto operací lze nalézt v kapitole hlavní nabídka.

Posun zobrazení grafu

Pro posun zobrazené oblasti stiskněte pravé tlačítko myši ve vykreslovací oblasti grafu. Dokud tlačítko znovu neuvolníte, bude se zobrazovaná oblast pohybovat podle pohybů myši.

Přibližování a oddalování grafu

Pro přiblížení grafu použijte tlačítko přiblížit. Každé přiblížení je dvojnásobné oproti předchozímu. Maximální přiblížení je 16:1.

Pro oddálení grafu použijte tlačítko oddálit. Každé oddálení zmenší graf na polovinu oproti předchozímu. Maximální oddálení je 1:16.

Tlačítko obnovit (obnovit zobrazení) nastaví původní přiblížení (1:1) a posunutí oblasti (na pozici 0,0).

Úpravy viditelnosti prvků grafu

Jednotlivé prvky grafu lze libovolně zobrazovat a schovávat.

Vrcholy grafu lze schovat nebo zobrazit pomocí seznamu vrcholů, který se nachází v pravém spodním rohu okna aplikace. k ovládání viditelnosti slouží zaškrtávací boxy vedle každého stavu.

Hrany grafu nelze schovat nebo zobrazit přímo. Jsou viditelné právě a jen tehdy, pokud jsou viditelné oba vrcholy, které hrana spojuje.

Výpis stavu lze schovat nebo zobrazit pomocí tlačítka na pravé straně odpovídajícího vrcholu grafu. Lze je také schovat pomocí tlačítka "×" v jejich pravém horním rohu.

Popis změny stavu lze schovat nebo zobrazit kliknutím na odpovídající hranu grafu.

Vybírání prvků grafu

Aplikace umožňuje několik způsobů, jak toto provést. Hrany grafu nelze samostatně měnit a proto ji nelze ani vybírat.

- Vybrání vrcholu grafu – Vrcholy lze vybírat zvlášť v seznamu vrcholů. Pokud není vybraný vrchol v aktuálním zobrazení vidět, je na něm zobrazení vystředěno. Jakýkoliv předchozí výběr se zruší.
- Vybrání více prvků Po jednom – Kliknutím na libovolný prvek grafu:
  - Bez stisknutých modifikačních kláves jej vyberete. Pokud ještě není vybraný, zruší se jakýkoliv předchozí výběr.
  - Se stisknutou klávesou Ctrl jej přidáte do výběru
  - Se stisknutou klávesou Shift jej odeberete z výběru.
  - Se stisknutými klávesami Ctrl a Shift jejich výběr invertujete.
  - Vybrání více prvků najednou – Stiskem levého tlačítka myši v prázdné ploše grafu a zahájením tažení myši se zobrazí výběrový obdélník. Při uvolnění tlačítka jsou ovlivněny všechny prvky, které zasahují do oblasti obdélníku:
  - Bez stisknutých modifikačních kláves všechny prvky vyberete. Jakýkoliv předchozí výběr se zruší.
  - Se stisknutou klávesou Ctrl všechny prvky přidáte do výběru.
  - Se stisknutou klávesou Shift všechny prvky z výběru odeberete.
  - Se stisknutými klávesami Ctrl a Shift invertujete výběr všech prvků.
- Zrušení celého výběru – Kliknutím do prázdné plochy grafu bez modifikačních kláves se zruší aktuální výběr.

Posun prvků grafu

Existují dvě možnosti, jakými aplikace umožňuje posunování prvků.

- Posunování vybraných prvků. Pokud bez stisknutých modifikačních kláves stisknete levé tlačítko myši nad libovolným vybraným prvkem a započnete táhnutí, zobrazí se obrysy všech vybraných prvků. Tyto obrysy vyznačují novou pozici vybraných prvků. Po uvolnění tlačítka myši se prvky na označená místa přesunou. Označené ukotvené prvky se přesunou podle obrysu. Neoznačené se posunou podle jejich kotev.
- Posunování prvků podle souřadnice. Pokud stisknete levé tlačítko myši v liště globálního posunování a započnete táhnutí, zobrazí se čára. Tato čára představuje hranici, za kterou jsou všechny prvky, které se pohnou, nezávisle na tom, zda jsou vybrané. Posunou se o tolik a v tom směru, o kolik a kterým směrem se posunul kurzor myši před tím, než jej uvolníte. Horizontální lišta slouží k posunování prvků napravo od hranice v horizontálním směru, vertikální pak k posunování prvků pod hranicí

ve vertikálním směru. Ukotvené prvky se nepohnou samostatně, nýbrž podle jejich kotev.

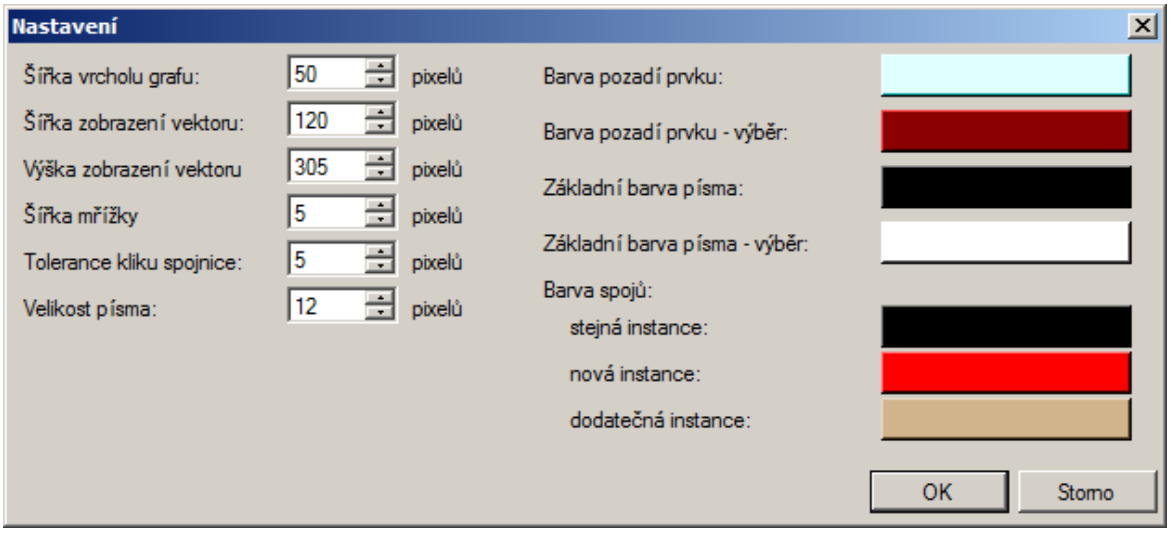
Při posunu se uplatňuje neviditelná mřížka, ke které se prvky přichytávají.

Přerušení výpočtu

Probíhající výpočet stavového prostoru je možné přerušit. Při výpočtu se zobrazí přerušovací dialog s jedním tlačítkem. Pokud se toto tlačítko zmáčkne, je výpočet přerušen. Při přerušení výpočtu zůstává graf stavového prostoru i síť CPN, ze které se počítal, ve stejném tvaru, jako před začátkem výpočtu. Pokud se tlačítko dialogu nezmáčkne, zmizí dialog ihned potom, co je stavový prostor dopočítán.

Nastavení aplikace

K nastavení aplikace slouží nastavovací dialog, který lze vyvolat z nabídky Možnosti



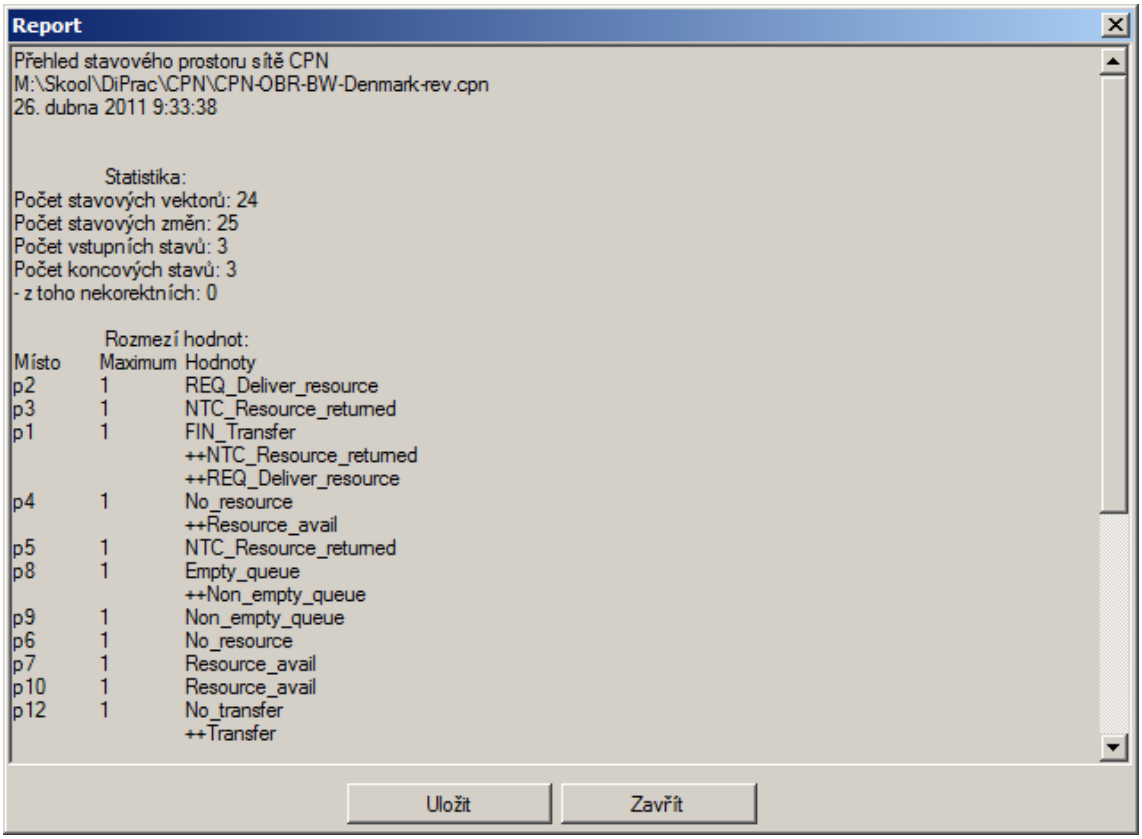
Obrázek 36 – Dialog nastavení aplikace

Lze nastavovat:

- Šířku vrcholů grafu. Výška vrcholu je odvozena z velikosti písma.
- Rozměry prvků výpisu stavu
- Šířku mřížky, ke které se prvky přichytávají.
- Maximální vzdálenost kurzoru od hrany, Pro kterou se kliknutí počítá jako kliknutí na hranu. Pokud je kurzor Při kliknutí od hrany blíž než tato vzdálenost, je kliknutí považováno za kliknutí na hranu.
- Velikost písma
- Barvy všech prvků grafu ve všech situacích.

Zpráva o stavovém prostoru

Zprávu o stavovém prostoru lze zobrazit v samostatném dialogu.



Obrázek 37 – Dialog zobrazení zprávy stavového prostoru

Dialog se skládá z needitovatelného textového pole, které obsahuje zprávu, tlačítka, které umožňuje uložení zprávy, a tlačítka, které dialog zavírá.

Zpráva obsahuje následující informace:

- Datum a čas, ve kterém byl stavový prostor vypočítán.
- Počet vrcholů grafu
- Počet hran graf
- Počet výchozích vrcholů grafu
- Počet terminálních vrcholů grafu
- Počet terminálních vrcholů grafu, které neodpovídají korektní specifikaci terminálního stavu sítí ABA-CPN. Tzn. vyskytují se v něm značky na místě, které není výstupní.
- Maximální počty značek na místech a jaké značky to byly. Značky z jednoho stavu jsou odděleny čárkou (","). Za seznamem je uvedeno číslo stavu, pro který tato situace nastala. Pokud bylo stavů, ve kterém místo

dosáhlo maximálního počtu značek více, jsou jednotlivé stavy oddělené novým řádkem.

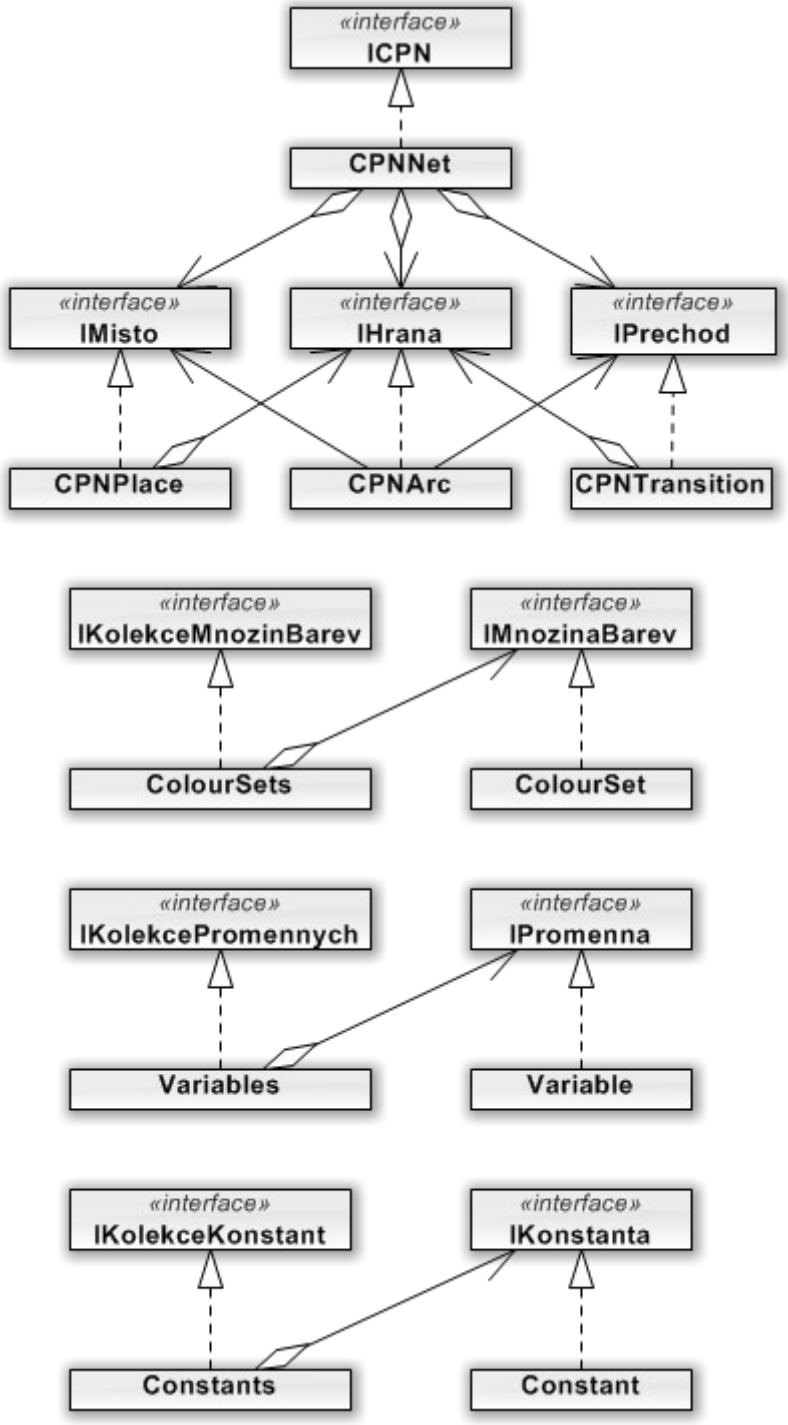
- Seznam koncových stavů, s tím, že nekorektní jsou označeny podtržítky.
- Seznam přechodů, které nebyly provedeny při žádném průchodu z libovolného počátečního stavu

Ostatní poznámky

Zde jsou sepsány poznámky a chyby, které se Při vývoji zatím nepodařilo odstranit.

- Po výpočtu občas zůstane viditelný přerušovací dialog, přestože už je graf vykreslený. Toto je způsobeno selháním koncové synchronizace. Dialog však zmizí, pokud přes něj přejedete myší nebo pokud jej učiníte aktivním oknem.
- Při přerušení se může objevit chyba, která hlásí, že se nepodařilo porovnat dva prvky v poli. Toto je způsobeno tím, že přerušení nastalo v nevhodný okamžik při porovnávání. Na další běh aplikace tato chyba vliv nemá a může být ignorována.

Příloha C: Struktura aplikace



Obrázek 38 – UML Diagram tříd a rozhraní definice sítě CPN

Na následujících diagramech jsou uvedeny UML diagramy tříd celé aplikace pro výpočet stavového prostoru sítě. Pro omezení velikosti diagramů nejsou zobrazeny operace ani atributy tříd a rozhraní. Všechny veřejné operace mají popis uvedený

v dokumentačních komentářích. V této příloze jsou pro přehlednost popsány pouze některé detaily implementace.

Třídy a rozhraní pro popis sítě CPN

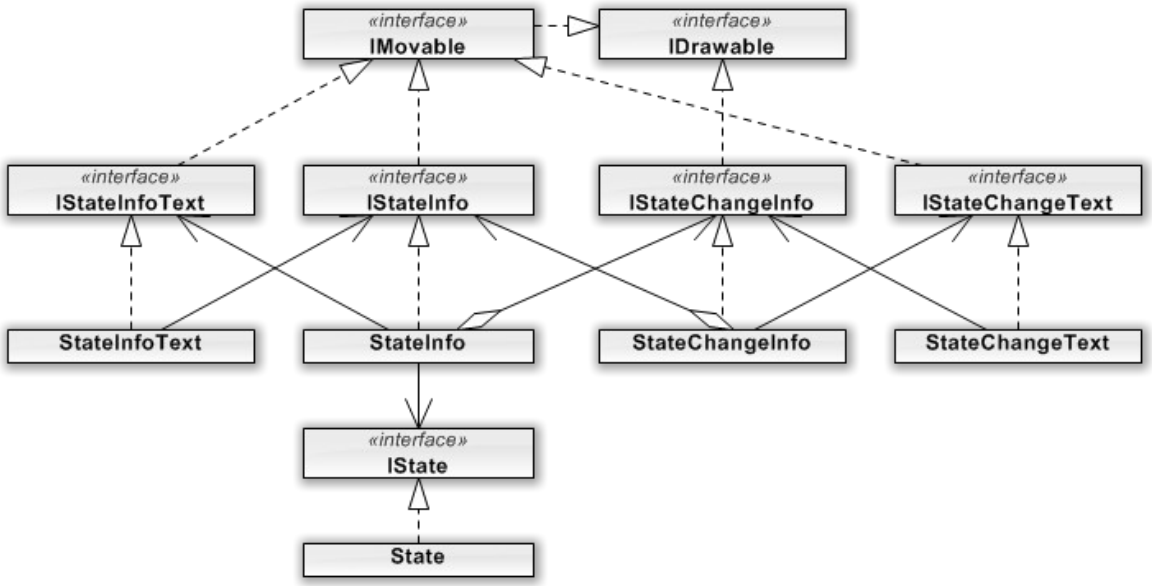
Tyto jsou zobrazeny na obrázku 38.

Pomocí rozhraní ICPN a třídy CPNNet je implementována síť CPN.

Pomocí rozhraní IMisto a třídy CPNPlace je implementováno místo sítě CPN. Pomocí rozhraní IPrechod a třídy CPNTransition je implementován přechod sítě CPN. Pomocí rozhraní IHrana a třídy CPNarc je implementována hrana sítě CPN. Je vidět, že hrany sítě a jednotlivé vrcholy jsou navzájem propojeny.

Pomocí rozhraní IPromenna a třídy Variable je implementována proměnná sítě CPN.Pomocí rozhraní IKonstanta a třídy Constant je implementována konstanta sítě CPN. Pomocí rozhraní IMnozinaBarev a třídy ColourSet je implementována množina barev sítě CPN. Rozhraní IKolekceMnozinBarev. IKolekceKonstant, IKolekcePromennych a odpovídající třídy ColourSets, Constants a Variables implemetují kolekce množin barev, konstant a proměnných sítě CPN.

Třídy a rozhraní grafu stavového prostoru



Obrázek 39 – UML diagram prvků grafu stavového porstoru

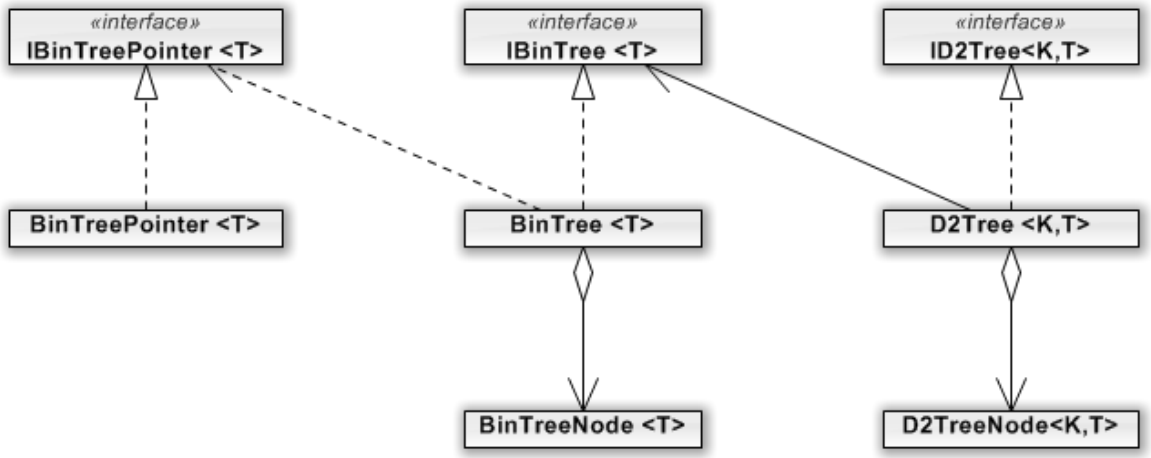
Tyto jsou zobrazeny na obrázku 39.

Rozhraní IStateInfo a třída StateInfo implementují vrchol grafu. Ten ve své implementaci využívá rozhraní IState implementované třídou State, ve které uchovává stavový vektor, a rozhraní IStateInfoText implementované třídou StateInfoText, které představuje prvek grafu, ve kterém se stavový vektor vypisuje.

Rozhraní IStateChangeInfo a třída StateChangeInfo implementují hranu grafu. Ta ve své implementaci využívá rozhraní IStateChangeText implementované třídou StateChangeText, které představuje prvek grafu, ve kterém se vypisují informace o změně stavu.

Všechny prvky kromě State jsou vykreslitelné, což definuje rozhraní Idrawable, a prvky IStateInfo, IStateInfoText a IstateChangeText jsou posunovatelné, což definuje rozhraní IMovable.

2D Strom nad binárním stromem



Obrázek 40 – UML diagram 2D stromu

Třídy pro implementaci 2D stromu jsou zobrazeny na obrázku 40.

2D strom je v této aplikaci vystavěn nad obecným stromem binárním.

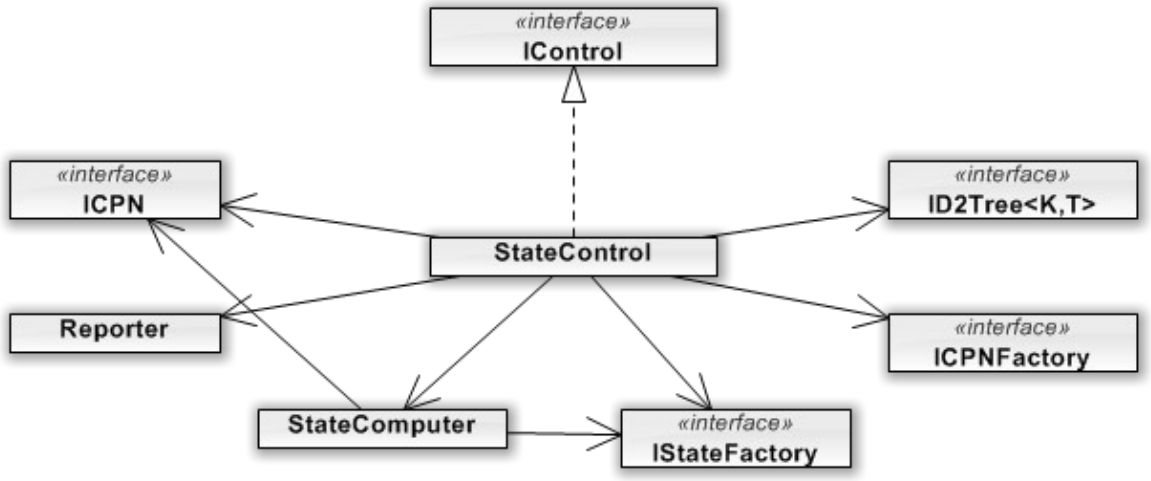
Při vkládání prvku do 2D stromu si automaticky vytvoří prvek, ve kterém uchovává data, a který vloží na místo do stromu binárního, které vyhledá. Obdobně se chová při vkládání binární strom, ale ten místo nehledá. Místo toho mu je řečeno, zda má prvek vložit jako levého nebo pravého potomka prvku, který je vybrán jako aktivní.

Pro operaci vyhledávání rozsahu klíčů si 2D strom potřebuje pamatovat místa větvení. Pro tento účel využívá rozhraní IBinTreePointer. Jedná se sice o prázdné rozhraní, ale skrývá se za ním privátní třída BinTreePointer, kterou vrací binární strom jako ukazatel na aktuálně vybraný prvek. Třída je privátní z toho důvodu, aby nemohlo dojít k jejím vnějším úpravám.

Tohoto rozhraní je využíváno i při operaci odebírání prvku ze 2D stromu, kdy je v podstromu odebíraného prvku hledán nejvhodnější prvek, který by odebíraný prvek nahradil.

Třídy vrstvy control

Tato vrstvy aplikace se stará o ovládání, jež souvisí s prvky vrstvy model, tedy prvky sítě, grafu apod. UML diagram vrstvy je zobrazen na obrázku 41.

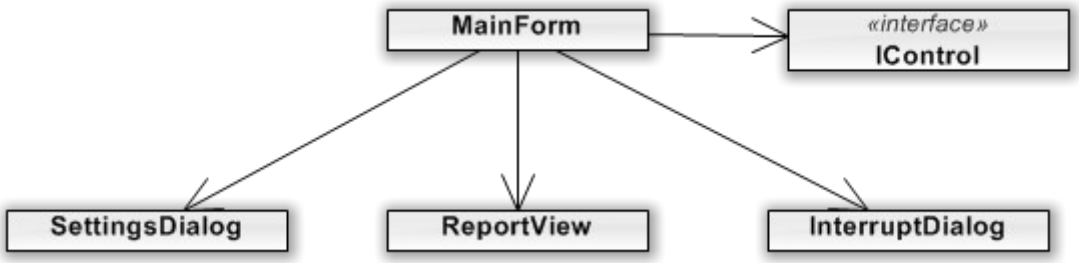


Obrázek 41 – UML diagram vrstvy control

Pro přístup z vyšší vrstvy slouží rozhraní IControl. Toto rozhraní implementuje třída StateControl. Ke třídě StateControl jsou přiřazeny další dvě třídy, které pomáhají s výpočtem, a sice třída StateComputer, která se stará o samotný výpočet a tvorbu prvků grafu, a třída Reporter, která se stará o generování zprávy o stavovém prostoru.

Třída má dále odkazy na síť CPN, se kterou se pracuje, a předává ji třídě StateComputer pro výpočet. Dále obsahuje odkaz na 2D strom, který obsahuje pohyblivé prvky grafu, a seznam hran grafu. Pro tvorbu sítě navíc obsahuje odkaz na rozhraní ICPNFactory, které obsahuje předpis tvorby sítě, a pro výpočet stavového porstoru předává třídě StateComputer odkaz na rozhraní IStateFactory, které obsahuje předpis tvorby prvků grafu stavového prostoru. Toto je navrženo podle návrhového vzoru abstraktní továrny.

Třídy vrstvy view



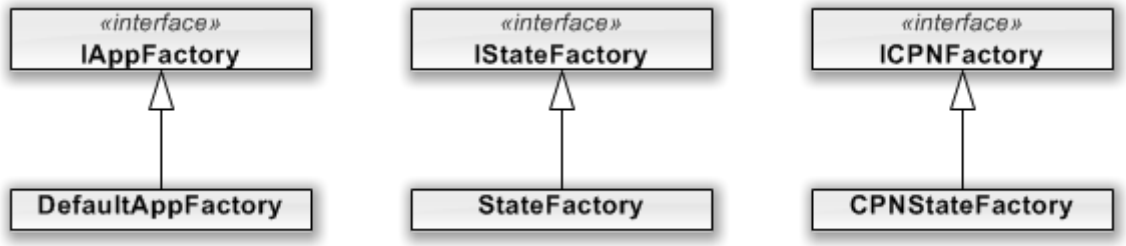
Obrázek 42 – UML diagram vrstvy view

Tato vrstvy aplikace se stará o zobrazování a ovládání, jež souvisí s prvky formulářů. UML diagram vrstvy je zobrazen na obrázku 42.

Vrstvy obsahuje hlavní třídu MainForm, jejíž event handlery působí jako spouštěče operací ostatních tříd a vrstev.

Do vrstvy dále patří další tři formuláře. SettingsDialog pro formulář nastavení, ReportView pro zobrazení zprávy o stavovém prostoru a InterruptDialog pro přerušení probíhajícího výpočtu.

Tovární třídy



Obrázek 43 – Tovární třídy

V aplikaci je definice tříd za rozhraními na mnoha místech provedena pomocí návrhového vzoru abstraktní továrna. Továrny v aplikaci je možné vidět na obrázku 43.

Třídy rozhraní ICPNFactory definují, jaké objekty se tvoří pro definici sítě. Třídy rozhraní IStateFactory určují, jaké objekty se tvoří pro definici grafu stavového prostoru. Třídy rozhraní IAppFactory určují, jaká se použije třída implementující rozhraní IControl a jaké třídy implementují rozhraní zbývajících dvou továren.

Další třídy

Další třídy mají podpůrnou funkci. Z důvodu přehlednosti a protože jsou jednoduché nejsou zahrnuty do UML diagramů.

Třída DBPanel představuje panel se zapnutým doublebufferingem, na který se vykresluje graf stavového prostoru.

Třída Pair je obecný kontejner pro dvojici prvků. Tyto prvky spolu nemusí souviset, v aplikaci však je použita pro uchování dvojice souvisejících prvků.

Třída Text je třída, která uchovává popis prvků sítě CPN. V aplikaci pro výpočet stavového prostoru je použita pouze pro kontrolu popisu hran při verifikaci sítě.

Třídy DruhyHrany, DruhMista a DruhPřechodu jsou výčtové typy, které slouží k určování, jakého typu jsou hrany, místa a přechody sítě CPN.

Třída PrefixChangeInfo je výčtový typ který udává, jak se zachovala nebo změnila instance formuláře zprávy při provedení přechodu.

Statická třída VerifStateSpace slouží k verifikaci sítě CPN.

Statická třída StateGraphicSettings slouží ke grafickému nastavení, podle kterého se zobrazují prvky grafu. Obsahuje nastavení velikostí, písma a barev.

Privátní třída NetOrigin formuláře MainForm je výčtový typ, který uchovává informaci, ze kterého zdroje přišla poslední síť. Ovlivňuje chování aplikace při volbě přepočítání grafu stavového prostoru.

Příloha D: Obsah přiložených souborů

K práci je přiloženo CD, jež obsahuje elektronickou verzi této práce ve formátech PDF a ODT a následující adresáře:

- src, který obsahuje projekt se zdrojovými kódy aplikace editoru a aplikace pro výpočet stavového prostoru,
- bin, který obsahuje zkompilovaný program z výše zmíněných zdrojových kódů a složku s HTML verzí nápovědy,
- examples, který obsahuje tři testované sítě uvedené v příloze A (soubory cpn) a uložené vypočítané stavové prostory (soubory stsp).