UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2011 Petr Nejman

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Interpret podtřídy barvené Petriho sítě

Petr Nejman

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. Petr NEJMAN**
Osobní číslo:	**I09375**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Interpret podtřídy barvené Petriho sítě**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

* Cílem diplomové práce je návrh a realizace interpreta ABA-CPN včetně vizualizace evoluce tohoto typu sítě. * V úvodní části práce je nutné provést přehled problematiky barvených Petriho sítí a její speciální podtřídy ABA-CPN. * Interpret ABA-CPN využívá jako vstup XML-soubor, který je standardně využíván nástrojem CPN Tools.

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 29. 7. 2011

Petr Nejman

PODĚKOVÁNÍ

Rád bych na tomto místě poděkoval především svému vedoucímu práce prof. Ing. Antonínu Kavičkovi, Ph.D. za jeho rady v teoretické oblasti.

ANOTACE

Práce se zabývá částí problematiky v ABAsim architektuře. Vysvětluje některé potřebné pojmy z oblasti simulací a oblasti Petriho sítí, ale hlavně je věnována podtřídě barvené Petriho sítě tzv. ABA-CPN, která je navržena a použita jako rozhodovací logika manažerů agentů. Cílem práce je pro tuto podtřídu vytvořit interpreta, který dokáže provést na této síti evoluci.

KLÍČOVÁ SLOVA

simulace, ABAsim architektura, barvená Petriho síť, ABA-CPN

TITLE

Interpreter of a subclass of coloured Petri net

ANNOTATION

The work deals with the issues in part ABAsim architecture. Explains some necessary concepts from domain of the simulations and the Petri nets, but is mainly devoted to the subclass of coloured Petri net so-called ABA-CPN, which is designed and used as a decision logic of agents' managers. The goal of a work is to create an interpreter, which can perform evolution on this net.

KEYWORDS

simulation, ABAsim architecture, coloured Petri net, ABA-CPN

Obsah

Seznam obrázků

1. Úvod

Jednou z možných a používaných architektur simulačních modelů je architektura označovaná jako ABAsim (agentově orientovaná architektura simulačních modelů). Tato architektura, jak už název napovídá, je složena z tzv. agentů, kteří mezi sebou mohou komunikovat. Ústředním článkem každého agenta je manažer, který má na starosti rozhodování. A právě jednou z možností, jak vytvořit rozhodovací logiku, je použití specielně navržené podtřídy barvené Petriho sítě ABA-CPN. Cílem této práce je vytvořit interpreta, který dokáže mimo jiné hlavně realizovat evoluci tohoto typu sítě.

Tato diplomová práce má následující hlavní cíle: Vytvořit editor pro ABA-CPN, jenž dokáže pracovat s xml soubory běžně používané nástrojem CPN Tools. V editoru bude možná verifikace ABA-CPN podle vysvětlené definice. Dalším cílem je vytvoření interpreta ABA-CPN, který dokáže graficky znázornit evoluci sítě. Tento interpret musí mimo barev značek sledovat i jejich instance. Posledním cílem je propojení vytvořeného editoru s diplomovou prací, na které pracoval Bc. Jiří Engliš, zabývající se výpočtem stavového prostoru ABA-CPN.

V této diplomové práci bude postupně vysvětlena potřebná teorie z oblastí Petriho sítí a ABAsim architektury. Dále bude detailně vysvětlena podtřída ABA-CPN, evoluční pravidla a konvence pro popis sítě. Další část se věnuje návrhu a vysvětlení některých částí vytvořeného softwaru, jako je např. verifikace sítě. V poslední části bych se chtěl věnovat testování vytvořeného produktu na demonstračních příkladech.

2. ABAsim architektura

Tato kapitola je věnována definici ABAsim architektury, což je jedna z možných architektur simulačních modelů. Budou zde vysvětleny základní stavební kameny této architektury a také její fungování. Nejprve jsou zde ale uvedeny a vysvětleny potřebné pojmy z oblasti modelování a simulace. Tato kapitola je inspirována citovaným zdrojem [2].

2.1. Základní pojmy z oblasti modelování a simulace

Pro správné pochopení dalších částí této kapitoly je důležité definovat základní pojmy z oblasti modelování a simulace.

Abstrakce a systém

Abstrakci si můžeme představit jako vymezení pouze některých aspektů zkoumaného objektu, které jsou pro nás důležité, a ostatní aspekty zanedbáme. Nezanedbané aspekty jsou vybírány takovým způsobem, že jsou zvládnutelné. V oblasti modelování a simulace se abstrakci říká systém. Můžeme také říci, že na zkoumaných objektech jsou vymezovány systémy.

Okolí systému

Okolí zkoumaného objektu představuje okolní objekty reálného světa, které sice nejsou předmětem zkoumání, ale přesto jsou důležité pro jejich vztah ke zkoumanému objektu. Proto je důležité uvažovat i jejich existenci a vlastnosti. Abstrakce okolí zkoumaného objektu se nazývá okolí systému.

Statický a dynamický systém

Ve statickém systému se abstrahujeme od významu času. Naproti tomu je tu dynamický systém, který význam času nezanedbává. Čas je tu chápán ve smyslu klasické „newtonovské“ (nikoliv kvantové) fyziky. Časová existence dynamického systému je dána množinou okamžiků, ve které dynamický systém existuje.

Model

Modelem v oblasti modelování a simulace je myšlena určitá analogie mezi modelovaným systémem (originálem) a modelujícím systémem. Vztah obou systémů je dán tím, že každému prvku P_O originálu je přiřazen prvek P_M modelujícího systému. Každému atributu a_O prvku P_O je přiřazen atribut a_M prvku P_M, přičemž pro hodnoty atributů a_O a a_M je dána nějaká relace.

Simulační model

Pokud jsou modelovaný i modelující systém statické systémy, jedná se o statický model. Ale v oblasti simulace se uplatňují pouze tzv. simulační modely, pro které musí platit:

1. Modelovaný systém (originál) i jeho modelující systém jsou dynamické systémy.

2. Existuje zobrazení τ existence originálu do existence modelujícího systému. Je-li t_1 časový okamžik, v němž existuje modelovaný systém O, je mu přiřazen $\tau(t_1) = t_2$, v němž existuje modelující systém M, a tak je zobrazením τ přiřazen i stavu ${}^{O}S_1(t_1) = {}^{O}s_1$ systému O stav ${}^{M}S_2(t_2) = {}^{M}s_2$ systému M.

3. Mezi stavy ${}^{O}s_1$ a ${}^{M}s_2$ jsou splněny požadavky na vztahy mezi prvky a jejich atributy, jak již bylo uvedeno při definici modelu.

4. Zobrazení τ je neklesající. Pokud nastane stav ${}^{O}s$ originálu před nějakým jeho jiným stavem $\underline{{}^{O}s}$, pak stav, který odpovídá v modelujícím systému stavu ${}^{O}s$, nastane před stavem, který odpovídá stavu $\underline{{}^{O}s}$ (nebo mohou nastat současně). Tento požadavek vyjadřuje nutnost dodržovat vztahy kauzality z originálu i v modelujícím systému.

Můžeme tedy říci, že model váže dva systémy, jejich prvky i atributy a pokud se jedná o simulační model, tak i samotnou existenci obou systémů.

V praxi se pod pojmem model rozumí modelující systém. U simulačních modelů se běžně používají místo modelovaného a modelujícího systému pojmy simulovaný a simulující systém. Podobně jako tomu bylo v případě modelujícího systému se místo simulujícího systému používá termín simulační systém nebo simulátor.

Modelování

Modelování ve smyslu výzkumné metody/techniky představuje nahrazení zkoumaného systému (originálu) jeho modelujícím systémem (modelem) za účelem získání informací o originálu pomocí pokusů s modelem.

Simulace

Simulace je výzkumná technika/ metoda, jejíž podstatou je, podobně jako tomu bylo u modelování, nahrazení zkoumaného systému jeho simulujícím systémem s cílem získat pomocí pokusů se simulujícím systémem informace o původním systému. Aby se jednalo o simulaci, je důležité, aby zkoumaný i simulující systém byly dynamickými systémy.

Architektura simulačních modelů

Architektura simulačních modelů jako pojem je chápan dosti volně, proto jsou zde uvedeny alespoň dva aspekty, které architekturu simulačních modelů charakterizují:

- Základní „stavební kameny“, tj. základní strukturální funkčně-akční jednotky, s jejichž pomocí je simulační model vystavěn (např. událost, aktivita, proces, agent, atd.).

- Realizace alokace simulačního výpočtu na výpočetní jednotky (procesory resp. počítače): monolitická architektura realizuje simulátor na jednoprocesorovém počítači, paralelní resp. distribuovaná architektura je realizována na víceprocesorovém počítači, resp. na více počítačích v rámci počítačové sítě.

2.2. Popis ABAsim architektury

Základním stavebním kamenem ABAsim architektury je agent. Jelikož sama problematika agentů je pro tuto práci velice důležitá, je zde tento pojem detailně popsán a vysvětlen.

Základní popis agenta

Agent je zapouzdřený počítačový systém zasazený do nějakého okolí, který v něm pružně a autonomně působí za účelem plnění daného cíle, přičemž za jeho klíčové vlastnosti považujeme:

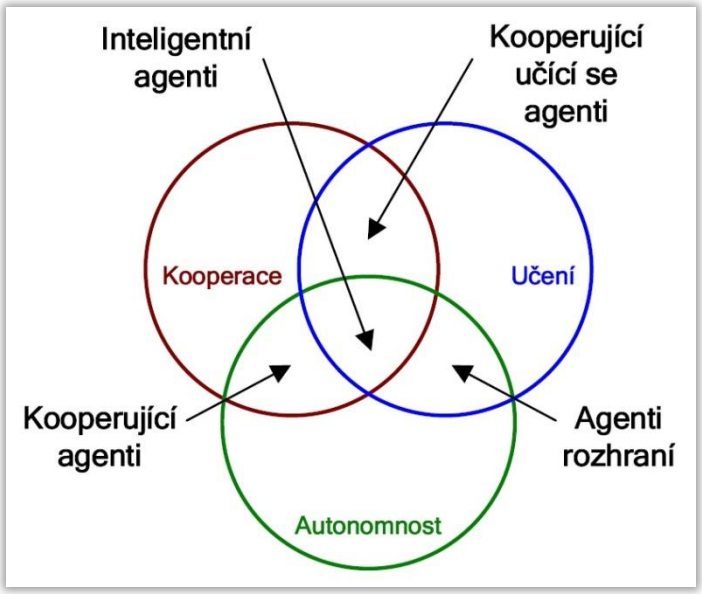
1. *autonomnost*, tj. agent může pracovat zcela samostatně bez vnějších vlivů na své chování a zcela řídit své výkony a kontrolovat vlastní vnitřní stav,
2. *společenské chování*, které se projevuje v komunikaci a interakci s okolím (jinými agenty nebo člověkem) pomocí jistého komunikačního mechanizmu,
3. *reaktivitu* tj. reakce na okolní podměty a
4. *iniciativnost*, tj. agent pouze nereaguje na okolní podměty, ale je také schopen vyvinout cíleným chováním iniciativu vůči svému okolí (podpora schopnosti učit se).

Pojem „zapouzdřený počítačový systém“ může znamenat čistě softwarovou realizaci agenta (softwarový agent), ale není to podmínkou. Agent může obsahovat i hardwarové komponenty.

Rozdělení agentů

Agenty můžeme dělit podle několika základních kritérií:

1. Podle *mobility*, tj. podle toho, zda mohou cestovat i v rámci počítačových sítí. Tyto agenty potom dělíme na *statické* („žijí pouze na jednom počítači“) a na *mobilní* agenty (mají možnost v rámci sítě cestovat).
2. Dalším kritériem je míra iniciativnosti agenta, kde podle tohoto kritéria můžeme rozlišovat *uvažující* (deliberativní) a *reaktivní* agenty. Uvažující agenti využívají interní symbolický model svého okolí, který je pro jejich chování určující. Pro reaktivní agenty ale platí jednodušší chování, které je postaveno na principu podnět→odezva, tj. reagují na aktuální stav svého okolí. Reaktivní agenti nedisponují žádným vnitřním symbolickým modelem svého okolí.
3. Jiná možná členění jsou postavena na různých kombinacích míry obsažení hlavních vlastností agenta, tedy na *autonomnosti*, *kooperativnosti* a na schopnosti *učit se* viz následující obrázek.



Obrázek 1 – Vlastnosti agenta [2]

Podle posledního typu dělení rozlišujeme *kooperující agenty*, *agenty rozhraní*, *kooperující učící se agenty* a *inteligentní agenty*. U uvedených typů agentů platí, že pro jejich chování jsou určující ty vlastnosti, které jsou obsaženy v průniku vlastností odpovídajícímu příslušnému typu agenta.

Mimo těchto tři základních kritérií můžeme najít i další kritéria, podle kterých by se dali agenti rozdělit. Například můžeme zařazení agentů určit přímo podle jejich poslání (př.: *internetový agent*, který bude působit v rámci internetu) nebo podle koncepčního přístupu, který byl při jejich tvorbě uplatněn.

Další běžně používané členění, které odráží nejpoužívanější typy agentů, může vypadat takto:

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

Reaktivní agenti

Vzhledem ke skutečnosti, že ABAsim architektura využívá právě reaktivní agenty, je zde tento typ agenta přiblížen o něco více.

Reaktivní agenti neobsahují žádný vnitřní symbolický model svého okolí, ale většinu svých akcí realizují jako odezvy na podněty svého okolí.

Základní rysy reaktivních agentů jsou:

1. Při tvorbě reaktivního agenta není vytvářen žádný plán jeho budoucího chování. Vzhledem k této skutečnosti jsou někdy tito agenti označování jako *situační agenti*, tedy agenti, kteří dopředu nevytvářejí žádné plány svého chování. Jejich činnost je tedy plně závislá na přítomnosti (okamžik odezvy na podnět).

2. Fyzická stavba agenta se sestává z modulů, které pracují samostatně, a každý z těchto modulů je určen pro specifickou práci (např. senzorická činnost, výpočty atd.). Mezi těmito moduly existuje pouze velmi omezená komunikace, a to poměrně na nízké úrovni. Žádný z agentů neobsahuje globální model, a proto chování agenta vyplyne až v rámci většího celku (z jeho okolí).

3. Činnost reaktivních agentů připomíná spíše činnost senzorických systémů.

„Inteligentní" agentové systémy mohou být postaveny na jednoduchých agentech (bez interních symbolických modelů), přičemž inteligence těchto systémů „se objeví" až v důsledku celého souhrnu situačních interakcí různých modulů.

Agenti v rámci ABAsim architektury

Každý agent má svůj úkol nebo cíl, který musí po dobu své existence plnit. Tento cíl může mít agent uložen od svého vzniku, nebo ho musí v průběhu své existence identifikovat (týká se především inteligentních agentů). Podle svého cíle potom každý agent realizuje svůj životní cyklus:

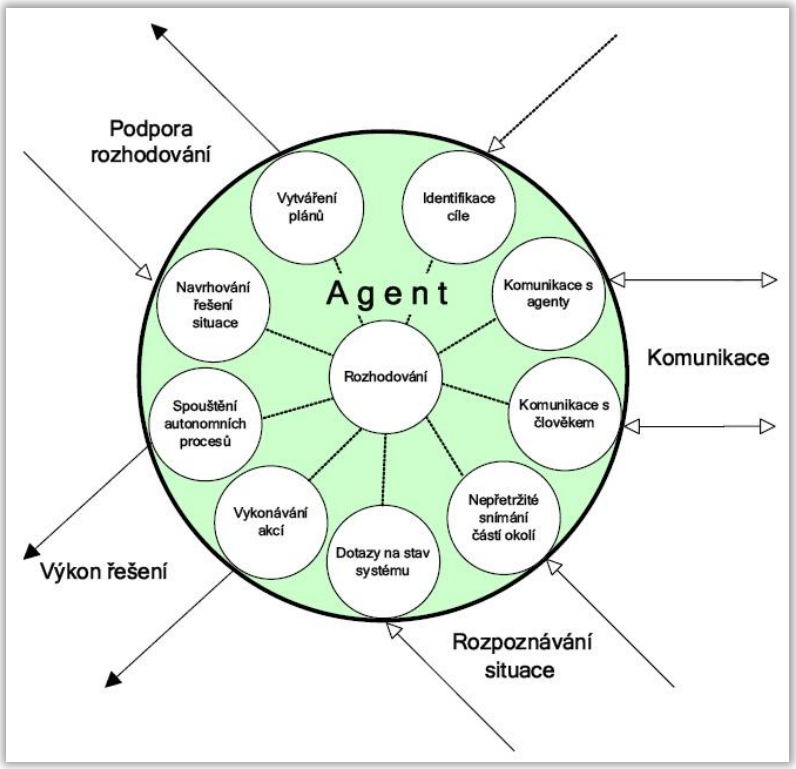
1. rozpoznání situace,

2. rozhodování o řešení a

3. výkon řešení.

Pro samotné rozhodování má agent k dispozici několik podpor rozhodování, jako je funkce návrhu řešení problémů a komunikace s okolím (s okolními agenty nebo s člověkem). Pokud agent rozpozná situaci, jejíž řešení ale nespadá pod jeho kompetenci, může pomoci ostatním agentům alespoň tak, že je o dané situaci upozorní.

Aby mohl agent vykonávat všechny tyto funkce, musí disponovat určitými schopnostmi. Pro možnost komunikace (nebo kooperace) s okolím musí mít agent schopnost komunikace. Pro podporu rozhodování disponuje schopností návrhu jednorázových řešení problémů a možností tvorby dlouhodobějších plánů řešení problémů. Tvorba dlouhodobějších plánů je sice už nad rámec reaktivních agentů, ale je zde chápána spíše pro pozdější využití. Pro senzorickou funkci agenta je potřeba schopnosti jednorázových dotazů na stav, stejně také možnost nepřetržitého snímání určitých částí systému. A nakonec pro možnost výkonu řešení je potřeba, aby byl agent schopen jednorázových akcí a také aby mohl jako výkon řešení spustit nějaký proces, který by tento výkon prováděl po nějakou dobu.



Obrázek 2 – Schopnosti agenta [2]

Výše zmíněné schopnosti agenta je vhodné rozdělit do skupin a každou ze schopností implementovat jako samostatnou komponentu. Tyto komponenty se potom mohou sdílet mezi agenty nebo si agent může vybírat mezi vícero alternativními komponentami, které plní stejnou úlohu.

Komponenty agenta

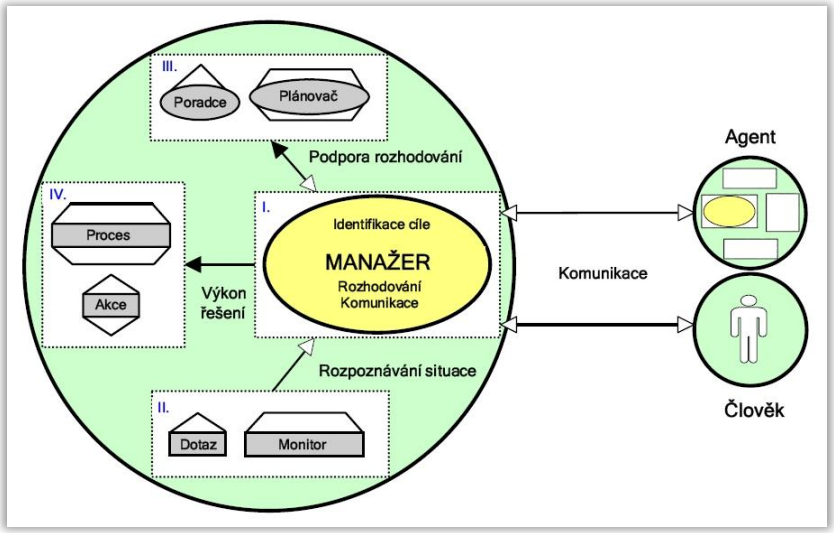
Ústřední komponentou agenta je *Manažer*. Má na starosti řízení, rozhodování a identifikaci cíle agenta. Dále také jako jediný má schopnost komunikace s dalšími interními komponentami agenta. Zbylé komponenty mezi sebou nekomunikují.

Další skupinou komponent jsou tzv. *senzory*. Spadají sem dvě podobné komponenty: *Dotaz*, který dokáže poskytnout informaci na popud manažera okamžitě a *Monitor*, který neustále snímá určitou část stavového prostoru simulátoru a manažerovi zasílá jen pro něj významné informace.

Třetí skupinou komponent jsou *řešitelé*. Mají za úkol dávat manažerovi návrhy na řešení různých situací. První komponentou je tzv. *Poradce*, který manažerovi na požádání „okamžitě“ poskytne návrh/návrhy řešení situace. Poradce je většinou nějaký optimalizační algoritmus nebo přímo člověk. Druhou možnou komponentou je *Plánovač*. Tato komponenta jde až za rámec reaktivních agentů, protože s jistým předstihem plánuje návrhy řešení možných problémů, ke kterým může v blízké budoucnosti dojít. Tyto návrhy jsou vytvářeny v podobě časových plánů, které plánovač bez vědomí manažera v jistých okamžicích aktualizuje. Pokud by tato komponenta odhalila nějaký vážný problém, který by měl manažer řešit, informuje ho o něm.

Poslední skupinou komponent jsou tzv. *efektory*. Tyto komponenty mají na starost vykonávání rozhodnutí manažera. Opět sem spadají dvě podobné komponenty, a to *Akce*, která provádí jednorázové změny v systému, a *Proces*, který samostatně provádí v různých časových okamžicích změny v systému.

V souvislosti s touto prací je důležité říci, že vnitřní logiku jednotlivých komponent lze vytvořit nejen pomocí programových rutin, konstruovaných pomocí některého z programovacích jazyků (*imperativně*), ale také může být realizována např. s uplatněním formalismů barvených Petriho sítí (*deklarativně*). Tato síť je poté v průběhu simulačního výpočtu používána interpretem příslušného typu sítě.



Obrázek 3 – Komponenty agenta [2]

Efektory, *senzory* i *řešitele* můžeme souhrnně nazývat *asistenty*, které podle povahy komponent můžeme dělit na:

- *kontinuální asistenty* (*Proces*, *Monitor* a *Plánovač*), jež svou činnost provádějí po určitou nenulovou dobu a
- *promptní asistenty* (*Akce*, *Dotaz* a *Poradce*), které svou činnost provedou v jednom okamžiku simulačního času.

Komunikační mechanizmus

Jak už bylo několikrát zmíněno, tak mezi agenty i v rámci jednoho agenta mezi manažerem a jeho asistenty probíhá určitá komunikace. Oba typy komunikace jsou v ABAsim architektuře realizovány výhradně pomocí zasílání zpráv. Tyto zprávy můžeme rozdělit do tří kategorií.

1. Zprávy, které jsou používány pro komunikaci mezi agenty resp. mezi jejich manažery.
   - *Notice* – Po odeslání této zprávy se neočekává odpověď.
   - *Request* – Jde o určitou žádost, na kterou se od adresáta očekává odpověď.

- *Response* – Tento typ zprávy se využívá právě jako odpověď na zprávu typu request a může být zaslána jen agentovi, který request zaslal.

2. Druhý typ zpráv slouží manažerovi, který je zasílá svým asistentům.

   - *Start* – Tento typ zprávy slouží k zahájení činnosti kontinuálního asistenta, který začne plnit svůj úkol ihned po přijetí této zprávy.

   - *Break* – Jedná se o jedinou zprávu, kterou může manažer ovlivnit samostatný průběh činnosti kontinuálního asistenta. Po přijetí této zprávy je činnost asistenta okamžitě ukončena. Tento typ zprávy se využívá je zřídka, a to jen v případech, kdy nemá smysl, aby tento asistent svou činnost dokončil.

   - *Execute* – Zprávou tohoto typu dává manažer pokyn svému promptnímu asistentovi, aby provedl svou činnost. Ten následně po dokončení činnosti vrací zprávu spolu s výsledkem své práce manažerovi.

3. Třetím a posledním typem zpráv posílají kontinuální asistenti informace svému manažerovi.

   - *Finish* – Jedná se o zprávu, kterou kontinuální asistenti povinně zasílají svému manažerovi ihned po ukončení své činnosti.

   - *Notice* – Tato zpráva může být zaslána kontinuálním asistentem kdykoliv. Označuje manažerovi nějakou situaci, která je pro něj významná.

   - *Hold* – Tento typ zprávy je používán jen kontinuálními asistenty, kteří si tuto zprávu posílají sami sobě. Jako jediná je opatřena tzv. *časovým razítkem*, které definuje čas doručení. Kontinuální asistent si tedy pošle tuto zprávu s posunutým časovým razítkem a jeho činnost je pozastavena až do doby opětovného doručení zprávy zpět. Právě zasíláním těchto zpráv je v ABAsim architektuře realizováno plynutí času.

3. Barvená Petriho síť

Tato kapitola je zaměřena na základní popis Petriho sítě, její základní elementy a grafickou reprezentaci. Dále je zde uvedena a i definice Barvené Petriho sítě, jejíž podtřídou se zabývá tato práce.

3.1. Petriho síť

Pojem Petriho síť (Petri net, PN) je označení pro širokou škálu diskrétních matematických modelů, které umožňují vlastními prostředky popisovat dynamiku modelovaného systému.

Existuje mnoho tříd Petriho sítí, které vznikly přidáním nových vlastností k základním prvkům a vlastnostem základní verze. Motivací jejich vzniku je získání větší modelovací mocnosti formalizmu, což je schopnost modelu vyjadřovat základní rysy modelovaného systému, nebo větší rozhodovací mocnost, která se vztahuje k existenci postupů analýzy vlastností modelů. [5]

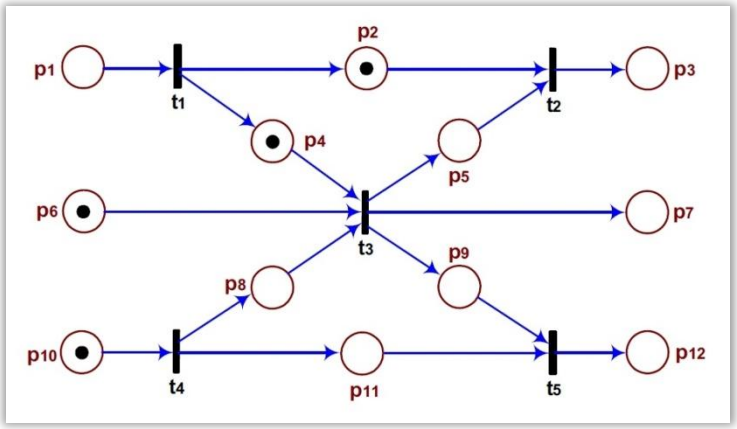
Modifikace buď rozšiřují, nebo zužují základní třídu Petriho sítí. Podle toho je řadíme mezi podtřídy, nebo rozšíření Petriho sítí. Podtřídy mají obvykle větší rozhodovací mocnost a rozšíření poskytují zvýšenou modelovací mocnost formalismu. [5]

Petriho sítě disponují dvěma disjunktními množinami elementů.

- První množinu tvoří *P-elementy* (*stavové elementy*, *místa*). V modelovaném systému jsou to většinou pasivní elementy (např. stavy, místa, zdroje, atd.).
- Druhou množinu tvoří tzv. *T-elementy* (*přechodové elementy*, *přechody*). Ty naproti tomu modelují aktivní elementy modelovaného systému (např. akce, události, posílání zpráv atd.).

Samotná evoluce Petriho sítě je realizována pomocí provádění přechodů. O proveditelnosti přechodu rozhoduje výhradně nejbližší okolí samotného přechodu (vstupní / výstupní místa a podmínky). Přechody, jejichž okolí je disjunktní, se mohou provádět nezávisle na sobě (souběžně, paralelně).

Petriho sítí se v průběhu evoluce pohybují značky (tokeny). Tyto značky jsou umístěny vždy jen na P-elementech a prováděním přechodů se z některých těchto P-elementů značky odstraňují a na jiné zas přidávají. Počáteční rozmístění značek je dáno inicializačním značením.



Obrázek 4 – Ukázka Petriho sítě

Grafická reprezentace Petriho sítě je následující: P-elementy jsou značeny kruhovými značkami a T-elementy obdélníkovými značkami. V grafu se dále objevují i orientované hrany, které definují okolí T-elementů. Tyto hrany jsou standardně značeny čarou, která je pro znázornění orientace ukončena šipkou. Značky jsou graficky znázorňovány pomocí teček. Mimo těchto čtyř základních grafických elementů se zde mohou objevit i různé popisky, identifikátory apod.

Ke každé grafické reprezentaci existuje i algebraická reprezentace, obsahující totožné informace o množině míst, přechodů, hran a případně i další informace.

3.2. Definice barvené Petriho sítě

Jedním z rozšíření základní Petriho sítě je Barvená Petriho síť (Colored Petri net, CPN). Každou síť, splňující následující definici, můžeme považovat za Barvenou Petriho síť. Definice byla přebrána ze zdrojů [5] a [6].

Devítice $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ je Barvená Petriho síť kde:

1. Σ je množina neprázdných typů, jinak také nazývaná *množina barev*.

   Množina typů Σ určuje hodnoty údajů, operací a funkcí, které je možné použít v síťových výrazech (např. hranové výrazy, strážní podmínky a inicializační výrazy). Předpokládáme, že každý typ obsahuje alespoň jeden prvek. Prvky mohou být jednoduché, nebo strukturované údajové objekty (např. n-tice, seznamy, záznamy).

2. P je konečná *množina míst*.

3. T je konečná *množina přechodů*.

4. A je konečná *množina hran*, pro kterou platí:

   $$P \cap T = P \cap A = T \cap A = \emptyset.$$

   Místa, přechody a hrany jsou popsány třemi množinami P, T a A, které musí být konečné a po dvojicích disjunktní. Požadavkem konečnosti množin se vyhneme mnohým problémům, jako například možnosti mít nekonečný počet hran mezi dvěma vrcholy.

5. $N: A \rightarrow (P \times T) \cup (T \times P)$ je *funkce vrcholů*.

   Funkce vrcholů N mapuje každou hranu na uspořádanou dvojici, ve které první prvek je zdrojový a druhý je cílový vrchol hrany. Oba vrcholy musí být různého druhu (jeden musí být místem a druhý přechodem). Tato definice také umožňuje mít více hran mezi vrcholy stejně uspořádané dvojice, což zvyšuje pohodlí při modelování.

6. $C: P \rightarrow \Sigma$ je *funkce barev*.

   Funkce barev C zobrazuje každé místo p na typ $C(p)$. To intuitivně znamená, že každá značka v místě p musí mít hodnotu typu $C(p)$.

7. G je *funkce strážních podmínek*, pro které platí:

$$G: T \rightarrow \{\forall t \in T: (Type(G(t)) = B \wedge Type(Var(G(t))) \subseteq \Sigma)\}.$$

   Funkce strážních podmínek G převádí každý přechod t do booleovského výrazu, ve kterém jsou všechny proměnné typu z množiny Σ. Při kreslení *CPN* se neuvádějí výrazy, ve kterých vyjde vždy výsledek *true*.

8. E je *funkce hranových výrazů*, pro které platí:

$$E: A \rightarrow \{\forall a \in A: (Type(E(a)) = C(p)_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma)\}.$$

   Funkce hranových výrazů E mapuje každou hranu na výraz typu $C(p)_{MS}$. To znamená, že každý hranový výraz se musí vyhodnotit do multi-množin nad typem přilehlého místa p. Diagram *CPN* může obsahovat i hranový výraz *expr* typu $C(p)$, což se považuje za zkrácení zápisu 1`(*expr*).

9. I je *inicializační funkce*, pro kterou platí:

$$I: P \rightarrow \{\forall p \in P: Type(I(p)) = C(p)_{MS}\}.$$

   Inicializační funkce I zobrazuje každé místo p na uzavřený výraz, který musí být typu $C(p)_{MS}$. Při kreslení *CPN* se vynechávají inicializační výrazy vyhodnocované jako ∅.

3.3. Chování barvené Petriho sítě

Následující popis chování barvené Petriho sítě je opět převzat z citovaných zdrojů [5] a [6].

Po definici struktury následuje popis chování *CPN*, pro které potřebujeme pro všechny přechody $t \in T$ a pro všechny dvojice vrcholů $(x_1, x_2) \in (P \times T) \cup (T \times P)$ tuto notaci:

- $A(t) = \{\forall a \in A | N(a) \in (P \times \{t\}) \cup (\{t\} \times P)\}$ – množina všech hran incidentních s přechodem t.
- $Var(t) = \{v | v \in Var(G(t)) \vee \exists a \in A(t): v \in Var(E(a))\}$ – množina všech proměnných, které se nacházejí ve strážní podmínce přechodu t a nebo na jeho incidentní hraně.
- $A(x_1, x_2) = \{a \in A | N(a) = (x_1, x_2)\}$ – množina hran mezi vrcholy x_1 a x_2.

- $E(x_1, x_2) = \sum_{a \in A(x_1, x_2)} E(a)$ – součet multi-množin z výrazů na hranách mezi vrcholy x_1 a x_2.

Vazbu přechodu t nazýváme takovou funkci b definovanou na $Var(t)$, pro kterou platí:

1. $\forall v \in Var(t): b(v) \in Type(v)$.
2. $G(t) < b >$

$B(t)$ označuje množinu všech vazeb pro t.

Značkový prvek je dvojice (p, c), kde $p \in P$ a $c \in C(p)$, vazbový prvek je dvojice (t, b), kde $t \in T$ a $b \in B(t)$. Množina všech značkových prvků se označuje TE a množina všech vazbových prvků se označuje BE.

Značení je multi-množina nad TE, pokud krok je neprázdná a konečná multi-množina nad BE. Počáteční značení M_0 je značení, které se získá vyhodnocením inicializačních výrazů:

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

Množina všech značení a kroků se označuje M, respektive Y.

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:

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

Proto se často značení reprezentuje funkcemi definovanými na množině P.

Krok Y je aktivovaný v značení M právě tehdy, pokud je splněna vlastnost:

$$\forall p \in P: \sum_{(t, b) \in Y} E(p, t)\langle b \rangle \leq M(p).$$

Nechť krok Y je aktivován při značení M. Když $(t, b) \in Y$, pak říkáme, že přechod t je aktivovaný při značení M pro vazbu b. Říkáme, že vazbový prvek (t, b) je aktivovaný při značení M, stejně jako přechod t. Když $(t_1, b_1), (t_2, b_2) \in Y$ a $(t_1, b_1) \neq (t_2, b_2)$, říkáme, že vazbové prvky (t_1, b_1) a (t_2, b_2) jsou souběžně aktivované, tak jako i t_1 a t_2. Když $|Y(t)| \geq 2$, pak říkáme, že přechod t je souběžně aktivovaný sám sebou. Když $Y(t, b) \geq 2$, pak říkáme, že vazbový prvek (t, b) je aktivovaný sám sebou.

Pokud je krok Y aktivovaný při značení M_1, může se vykonat, přičemž změní značení M_1 na jiné značení M_2, definované:

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

První suma se nazývá odstraněné značky a druhá se nazývá přidané značky. Říkáme, že M_2 je přímo dosažitelné z M_1 výskytem kroku Y, což se zapisuje:

$$M1[Y > M2.$$

Konečná posloupnost vykonávání přechodů je posloupnost značení a kroků:

$$M_1[Y_1 > M_2[Y_2 > M_3 \ldots M_n[Y_n > M_{n+1}$$

taková, že $n \in N$ a $M_i[Y_i > M_{i+1}$ pro všechny $i \in \{1, 2, \ldots, n\}$, M_1 je počáteční značení, M_{n+1} je koncové značení a n je délka.

Analogicky se nekonečná posloupnost vykonávání přechodů nazývá posloupnost značení a kroků:

$$M_1[Y_1 > M_2[Y_2 > M_3 \ldots$$

taková, že $M_i[Y_i > M_{i+1}$ pro všechny $i \geq 1$.

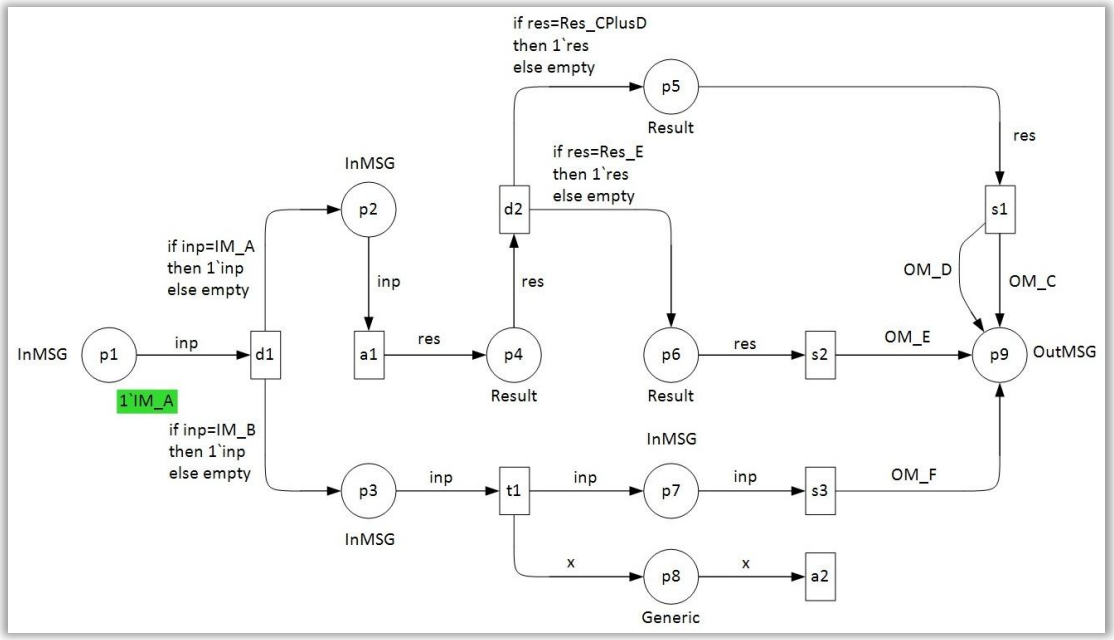
Značení M'' je dosažitelné ze značení M', existuje-li konečná posloupnost vykonání přechodů začínající v M' a končící v M''. Množina značení dosažitelných ze značení M' se označuje $[M' >$. Značení je dosažitelné, pokud patří do $[M_0 >$.

4. ABA-CPN

Z hlediska dosažení vysoké flexibility cílového simulačního modelu se jeví jako klíčové věnovat zvýšenou pozornost popisu logiky řídících komponentů agentů (tj. manažerů). Pro formalizaci této logiky byla vyvinuta původní podtřída nehierarchické barvené Petriho sítě (označované jako ABA-CPN), jejíž specifikaci se budeme věnovat v této kapitole.

4.1. Definice ABA-CPN

Některé části definice uvedené v této kapitole, budou pro snazší pochopitelnost demonstrovány na následujícím ilustračním obrázku. Definice uvedená v této podkapitole, je převzata ze zdroje [4].



Obrázek 5 – Ilustrační síť ABA-CPN

ABA-CPN představuje podtřídu *CPN* = (*Σ*, *P*, *T*, *A*, *N*, *C*, *G*, *E*, *I*), která vyhovuje následujícím podmínkám:

1. *Σ* je množina neprázdných typů nazývaných *množina barev*.

   Ilustrační síť disponuje následujícími množinami barev *Σ* = {*InMSG*, *Result*, *Generic*, *OutMSG*}. Množina *InMSG* sestává ze dvou prvků (barev) *IM_A* a *IM_B*, množina *OutMSG* obsahuje čtyři elementy: *OM_C*, *OM_D*, *OM_E* a *OM_F*. Množina barev *Result* disponuje dvěma prvky *Res_CPlusD* a *Res_E* a konečně množina *Generic* zahrnuje pouze jeden prvek *e*.

2. Konečná *množina míst P* = {P_{in}} ∪ {P_{out}} ∪ P_S, P_{in} je *vstupní místo*, P_{out} *výstupní místo* a P_S se skládá z *interních míst*, přičemž tři uvedené množiny jsou vzájemně disjunktní.

   Množina míst *P* je rozdělena do podmnožin z důvodu lepší rozlišitelnosti specifických typů míst. Výskyt značky ve vstupním místě p_{in} odpovídá přijetí

vstupní zprávy agenta, značka ve výstupním místě p_{out} odráží připravenost výstupní zprávy agenta k odeslání příslušnému jinému agentovi. V ilustrační síti platí: *vstupní místo* $p_{in} = p_1$, *výstupní místo* $p_{out} = p_9$ a *interní místa* soustřeďuje $P_S = \{p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.

3. Konečná *množina přechodů* $T = T_D \cup T_A \cup T_S \cup T_B$, $T \neq \emptyset$, kde elementy T_D jsou označovány jako *rozhodovací přechody*, elementy T_A jako *asistenční přechody*, elementy T_S jako *odesílací přechody* a elementy T_B jako *standardní přechody*. Všechny čtyři uvedené množiny jsou vzájemně disjunktní a $T \cap P = \emptyset$.

   Množina přechodů T je složena ze čtyř podmnožin. *Rozhodovací přechody* (zahrnuté v T_D) představují body podmínkového větvení (přechody d_1 a d_2 na obrázku 5), *asistenční přechody* (obsažené v T_A) odrážejí interní výkonné komponenty/asistenty agentů (a_1, a_2), *odesílací přechody* (prvky množiny T_S) modelují zasílání zpráv jiným komponentům, resp. agentům (s_1, s_2, s_3). *Standardní přechody* (uchovávané v T_B) nejsou spojeny s žádnou specifickou interpretací vzhledem k používané architektuře simulačního modelu (příkladem standardního přechodu je t_1).

4. A představuje konečnou *množinu hran*, pro kterou platí: $P \cap A = T \cap A = \emptyset$.

5. N je *vrcholová funkce* definovaná z A do $(P \times T) \cup (T \times P)$.

6. C představuje *funkci barev* definovanou z P do Σ.

   Příklady funkcí barev jsou $C(p_1) = InMSG$ a $C(p_6) = Result$.

7. E je *funkce hranových výrazů* definovaná z A do *množiny hranových výrazů*.

8. Hrany disponují následujícími specifikacemi:

   a. Neexistuje dvojice hran a_1, a_2, $a_1 \neq a_2$, pro kterou $p(a_1) = p(a_2) \wedge t(a_1) = t(a_2) \wedge ((N(a_1) \in T \times P \wedge N(a_2) \in P \times T)$, kde $p(a)$ představuje místo a $t(a)$ reprezentuje přechod z $N(a)$,

   b. Každá hrana $a \in A$ náleží do jedné z následujících kategorií:

      - Jestliže $t(a) \in T_D \wedge N(a) \in T \times P$, pak je a nazývána jako rozhodovací hrana a příslušný hranový výraz $E(a)$ obsahuje podmínku pro variantní větvení.

      - Pro všechna $t(a) \notin TD$ a všechna $t(a) \in T_D$: $N(a) \wedge P \times T$, představuje a jednu z následujících množin: *konstantní hrana*, je-li $E(a)$ složeno pouze z jedné konstanty, nebo *elementární proměnná hrana*, jestliže $E(a)$ obsahuje pouze jednu proměnnou.

   Specifikace hran nepřipouští konstrukci vlastních cyklů. Přípustné konstrukce hranových výrazů lze rozdělit do dvou kategorií. Hranové výrazy příslušející hranám vycházejícím z rozhodovacích přechodů (*rozhodovací hrany*) představují podmínkové výrazy. V rámci ilustrační sítě se jedná o hrany (d_1, p_2), (d_1, p_3), (d_2, p_5) a (d_2, p_6). Hrany, jejichž počátek je v jiných vrcholech než rozhodovacích přechodech, disponují hranovými výrazy obsahujícími buď pouze jednu konstantu (*konstantní hrany*), nebo jednu

proměnnou (*elementární proměnné hrany*). Na demonstračním obrázku jsou následující elementární proměnné hrany (p_1, d_1), (p_2, a_1), (a_1, p_4), (p_4, d_2), (p_5, s_1), (p_6, s_2), (p_3, t_1), (t_1, p_7), (p_7, s_3), (t_1, p_8), (p_8, a_2). Všechny zbývající hrany jsou klasifikovány jako *hrany konstantní*.

9. G představuje *funkci strážní podmínky* definovanou z T do *množiny výrazů strážních podmínek*. Pro ABA-CPN platí: $\forall t \in T$: $G(t) = \emptyset$, tj. přechody nedisponují žádnými strážními podmínkami.

10. Prvky množin P a T disponují následujícími přídavnými vlastnostmi:

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

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

    c. $\forall p \in P_S$: $(\exists\, a \in A$: $N(a) = (t, p), t \in T) \wedge (\exists_1\, a \in A$: $N(a) = (p, t), t \in T)$,

    d. $\forall t \in T_B$: $\exists\, a \in A$: $N(a) = (p, t), p \in P$,

    e. $\forall t \in T \setminus T_B$: $\exists_1\, a \in A$: $N(a) = (p, t), p \in P$,

    f. $\forall t \in T_D \cup T_S$: $\exists_1\, a \in A$: $N(a) = (t, p), p \in P$,

    g. $\exists_1\, t \in T_D$: $\exists_1\, a \in A$: $N(a) = (p_{in}, t)$, kde t je označován jako *vstupní přechod*,

    h. $t \in T$: $(\exists\, a \in A$: $N(a) = (t, p_{out})) \vee (\neg\exists\, a \in A$: $N(a) = (t, p), p \in P)$ je nazýván jako *výstupní přechod*,

    i. $t \in T$: $(\neg\exists\, a \in A$: $N(a) = (p_{in}, t)) \wedge (\neg\exists\, a \in A$: $N(a) = (t, p_{out})) \wedge (\exists\, a \in A$: $N(a) = (t, p), p \in P)$ je označován jako *interní přechod*.

    Vlastnosti *a*, *b*, *c* se týkají vrcholů-míst: do *vstupního místa* nejsou zaústěny žádné hrany a právě jedna hrana z něj vychází (místo p_1 v ilustrační síti); *výstupní místo* je incidentní pouze s hranami do něj zaústěnými (místo p_9); všechna ostatní místa jsou incidentní s alespoň jednou vstupní hranou a právě jednou výstupní hranou.

    Vlastnosti *d* až *i* charakterizují přechody. Všechny přechody jsou incidentní alespoň s jednou vstupní hranou. Do *rozhodovacích*, *asistenčních* a *odesílacích přechodů* je zaústěna právě jedna hrana, *standardní přechody* mohou být incidentní s více vstupními hranami. Alespoň jednou výstupní hranou musí disponovat *rozhodovací* a *odesílací přechody* (např. d_1, d_2, s_1, s_2 a s_3 na demonstračním obrázku), což nemusí platit pro ostatní typy přechodů (např. z přechodu a_2 nevychází žádná hrana). *Vstupní přechod* je právě jeden a jako jediný ze všech přechodů leží ve výstupním vrcholovém okolí *vstupního místa* (přechod d_1 do něhož vede hrana ze vstupního místa p_1). *Výstupní přechody* jsou buď spojeny hranou s *výstupním místem*, nebo mají prázdné výstupní vrcholové okolí (přechody s_1, s_2, s_3 a a_2). Všechny ostatní přechody jsou nazývány jako *interní* (přechody d_2, a_1 a t_1).

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

12. I je *inicializační funkce* definovaná z P do *množiny uzavřených výrazů* (zmíněné výrazy specifikují multimnožiny barev).

13. *Množina přípustných inicializačních značení* $M_0 \subset I(P)$ je definována následovně: $M_0 = \{{}^jM_0 | j = 1, 2, \ldots, |C(p_{in})|\}$, kde ${}^jM_0$ označuje j-té inicializační značení, pro které platí: $\forall p \in P$: $|{}^jM_0(p)| = 1$, jestliže $p \neq p_{in}$; pro $i \neq j$ platí: ${}^iM_0(p) \neq {}^jM_0(p)$.

    Značky odlišných barev odrážejí v rámci ABA-CPN odlišně vyplněné formuláře zpráv, které jsou v rámci architektury ABAsim využívány pro komunikační účely. Přípustné inicializační značení ABA-CPN připouští výskyt právě jedné značky v rámci celé sítě umístěné ve vstupním místě p_{in}; žádné z dalších míst sítě nedisponuje žádnou značkou. Tento stav odráží situaci, kdy vstupní zpráva (příslušného komponentu agenta) čeká ve vstupním místě na své zpracování, přičemž daná síť aktuálně žádné jiné zprávy nezpracovává. ABA-CPN je navržena pro separátní zpracování všech typů příslušných vstupních zpráv (reprezentovaných značkami z množiny barev vstupního místa $C(p_{in})$), proto množina přípustných inicializačních značení M_0 obsahuje tolik značení, kolik je druhů vstupních zpráv (tj. $|C(p_{in})|$) – ${}^jM_0$ označuje j-té inicializační značení. Na ilustračním obrázku je zobrazeno jedno přípustné inicializační značení: vstupní místo p_1 obsahuje jednu značku barvy *IM_A*, zatímco ostatní místa jsou prázdná. Tento stav reprezentuje situaci přijetí zprávy *IM_A* (*Input Message A*) příslušným komponentem agenta, který ji dále zpracuje.

4.2. Konvence pro popis

Pro volby identifikátorů jednotlivých proměnných a konstant v rámci hranových výrazů byly zavedeny určité konvence, jejichž pomocí může návrhář ABA-CPN ovlivnit tvorbu nových instancí značek (zpráv), které se v rámci sítě vyskytují. Veškeré konvence pro popis jsou převzaty ze zdroje [4].

Z důvodu provedení jednoznačné transformace ABA-CPN sítě do příslušných datových struktur v rámci simulačního programu a provádění korektní evoluce sítě specializovaným interpretem byly navrženy následující konvence pro označování elementů ABA-CPN:

- Místa jsou označována identifikátory p_i, pro $i = 1, \ldots, n$, kde $n = |P|$. p_1 značí *vstupní místo* a p_n *výstupní místo* sítě.
- Pro rozhodovací přechody jsou identifikátory d_i, pro $i = 1, \ldots, m$, kde $m = |T_D|$. Identifikátorem d_1 značíme *vstupní přechod*.
- Standardní přechody označujeme t_i, pro $i = 1, \ldots, k$, kde $k = |T_B|$.
- Asistenční přechody mají identifikátory a_i, pro $i = 1, \ldots, l$, kde $l = |T_A|$.
- Odesílací přechod nese identifikátory s_i, pro $i = 1, \ldots, r$, kde $r = |T_S|$.
- Konstrukce hranových výrazů pro konstantní a elementární hrany sestává pouze z jediného symbolu, který je dán deklarovanou proměnnou, konstantou nebo elementem deklarované množiny barev.

- Hranové výrazy rozhodovacích hran jsou zapisovány ve formě „if *var = const* then *případ1* else *případ2*“, kde *var* značí proměnnou, *const* konstantu, odpovídající typem proměnné *var*, a *případ1* resp. *případ2* označuje proměnnou, konstantu, nebo klíčové slovo *empty*.

Konvence pro značení proměnných a konstant v hranových výrazech sledují cíl, aby bylo návrháři sítě umožněno ovlivňovat evoluci sítě vzhledem k „datovému obsahu“ příslušné značky (v našem případě obsahu příslušného formuláře zprávy). Využívání níže popsaných konvencí navíc návrháři tedy umožňuje řídit „průchod“ instancí jednotlivých značek danou sítí v průběhu její evoluce (tato koncepce jde samozřejmě za rámec standardní definice evoluce Petriho sítě, která je založena na zániku značek po jejich odebrání z míst ve vstupním vrcholovém okolí daného prováděného přechod a na vzniku nových značek, které jsou následně umístěny do míst ve výstupním vrcholovém okolí zmíněného přechodu).

Pro potřebu následujícího výkladu bude potřeba poukázat na určitý znak z řetězce. Pokud budeme mít řetězec *s*, tak jeho *i*-tý znak bude značen jako s^i.

Konvence pro popis proměnných a konstant v rámci výrazů hran incidentních k přechodu $t \in T$ jsou následující:

1. Při provádění přechodu $t \in T$ platí, že instance značky (případně její identická kopie) odebraná z místa $p \in P$: $N(a) = (p, t)$, $a \in A$ je následně umístěna do místa $q \in P$: $N(a) = (t, q)$, $a \in A$ za podmínek:

   - instance značky vyskytující se v místě *p* je z tohoto místa odebrána prostřednictvím konstanty resp. proměnné (jejíž identifikátor označme jako *input*) obsažené ve výrazu hrany (*p*, *t*),

   - umístění instance značky do místa *q* je realizováno prostřednictvím konstanty, resp. proměnné (jejíž identifikátor označme *output*) obsažené ve výrazu hrany (*t*, *q*),

   - pro zmíněné identifikátory konstant a/anebo proměnných platí: $input^1 = output^1$, tj. první znaky obou identifikátorů jsou totožné.

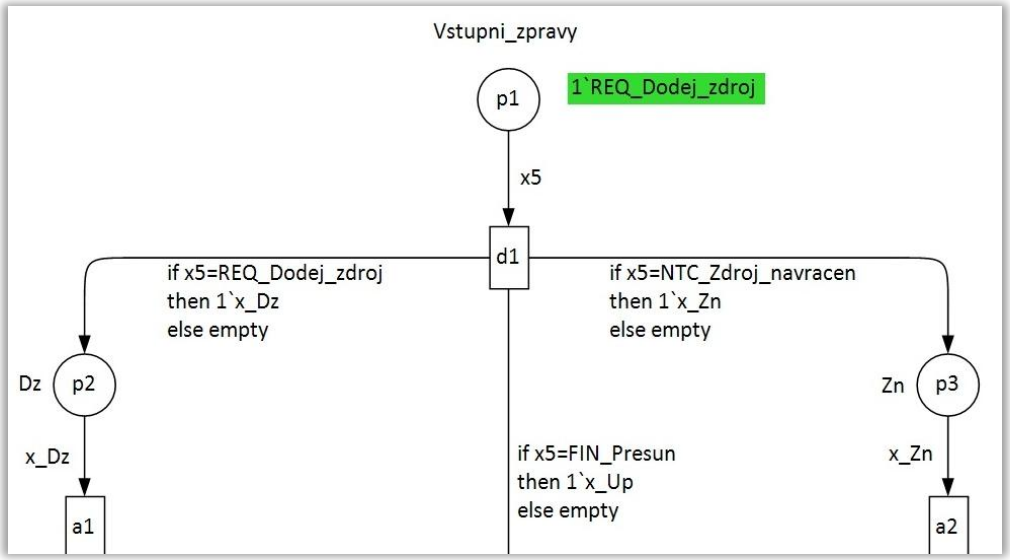
   Představená konvence umožňuje návrháři sítě (zahrnuté v rámci simulačního modelu založeného na architektuře ABAsim) určovat konkrétní „trajektorii průchodu“ instance značky (nesoucí specifická data) danou sítí. Tato vlastnost představuje rozšíření pravidel chování standardní barvené Petriho sítě. To znamená, že při provádění přechodu nemusí být vždy prováděny „dealokace“ (stávajících instancí) formulářů zpráv odebíraných z příslušných vstupních míst a „alokace“ (nových instancí) formulářů umísťovaných do výstupních míst přechodu, nýbrž mohou být příslušné instance formulářů zpráv zachovány.

2. Druhý znak identifikátoru (znakového řetězce) proměnné (nikoliv konstanty) z příslušného hranového výrazu se využívá pro určení dat, která jsou obsažena v rámci *j*-té položky dané značky *c*, což je využito zejména při provádění rozhodovacího přechodu. Provedení rozhodovacího přechodu $t \in T_D$ probíhá v následujících fázích:

- instance značky c vyskytující se v místě p ∈ P: N(a) = (p, t), a ∈ A je z tohoto místa odebrána prostřednictvím proměnné (jejíž identifikátor označme jako input) obsažené ve výrazu hrany (p, t),
- druhý znak řetězce input je využit ke stanovení hodnoty $j = int(input^2)$, kde funkce *int* transformuje znak $input^2 \in M$ (množina znaků '1', …, '9') na odpovídající celé číslo z intervalu ⟨1, …, 9⟩,
- získaný datový obsah *j*-té položky značky *c* (označme jej jako $val^{c,j}$) představuje konstantu, která je využita při provádění rozhodovacího přechodu *t* v rámci zpracování výrazů rozhodovacích hran (specifikovaných ve tvaru „if *var* = $val^{c,j}$ then *případ1* else *případ2*") vycházejících z tohoto přechodu.

3. Třetí znak identifikátorů proměnných se využívá pouze v případech, kdy v síti existuje více identifikátorů se stejnými prvními dvěma znaky, přičemž tyto identifikátory jsou spojeny se značkami odlišných barev.
4. Konvence pro značení identifikátorů konstant (vzhledem k jejich druhým a dalším znakům) jsou odlišné od přístupu uplatňovaného na identifikátory proměnných. Pro identifikátory konstant je uplatňována pouze konvence uvedená v bodě 1.

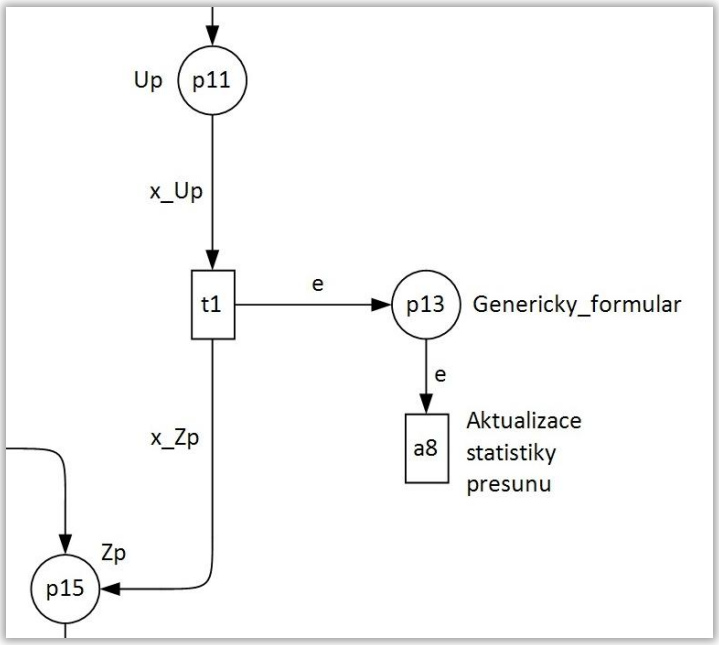
Na následujících obrázcích (výřezy z ABA-CPN sítě) budou předvedeny některé konvence pro tvorbu identifikátorů proměnných a konstant.



Obrázek 6 – Ilustrace1 pro popis konvence tvorby identifikátorů

Pro vstupní rozhodovací přechod d_1 platí, že hranové výrazy spojené se všemi jeho incidentními hranami jsou složené z konstant a proměnných, jejichž identifikátory mají tu vlastnost, že jejich první znak je roven '*x*' ('*x5*', '*x_Dz*', '*x_Up*', '*x_Zn*'). Proto je zpráva (reprezentovaná instancí značky odebrané z místa p_1) je následně přesunuta do příslušného výstupního místa *q*. Druhý znak identifikátoru proměnné spojené s elementární proměnnou hranou (p_1, d_1) se rovná '5'. To znamená, že interpret ABA-CPN rozhoduje o variantním větvení podle aktuálního obsahu páté datové položky příslušné zprávy (reprezentované

značkou *c* aktuálně odebranou z místa p_1). Například, pro případ $val^{c.5}$ = *REQ_Dodej_zdroj* je značka *c* přesunuta do místa p_2.



Obrázek 7 – Ilustrace2 pro popis konvence tvorby identifikátorů

Situace na přechodu t_1 je rozdílná: provedením přechodu není využita pouze instance značky odebrané z místa p_{11} (a následně přesunuté do místa p_{15}), nýbrž je vytvořena nová instance odlišného typu značky, která je umístěna do místa p_{13}. Toto chování vyplývá z aktuálních výsledků porovnání hodnot prvních znaků relevantních identifikátorů konstant (*inp* = '*x_Up*', *outa* = '*x_Zp*', *outb* = '*e*', tedy $inp^1 = outa^1$, $inp^1 \neq outb^1$).

5. ABA-CPN Editor

Tato kapitola je věnována aplikaci, která je součástí výsledků této diplomové práce. Nejprve je zde představen nástroj CPN Tools a jako alternativa editor, který je součástí výsledné aplikace. Dále jsou zde ukázány datové struktury využité v aplikaci a popsány jednotlivé aspekty verifikace sítě.

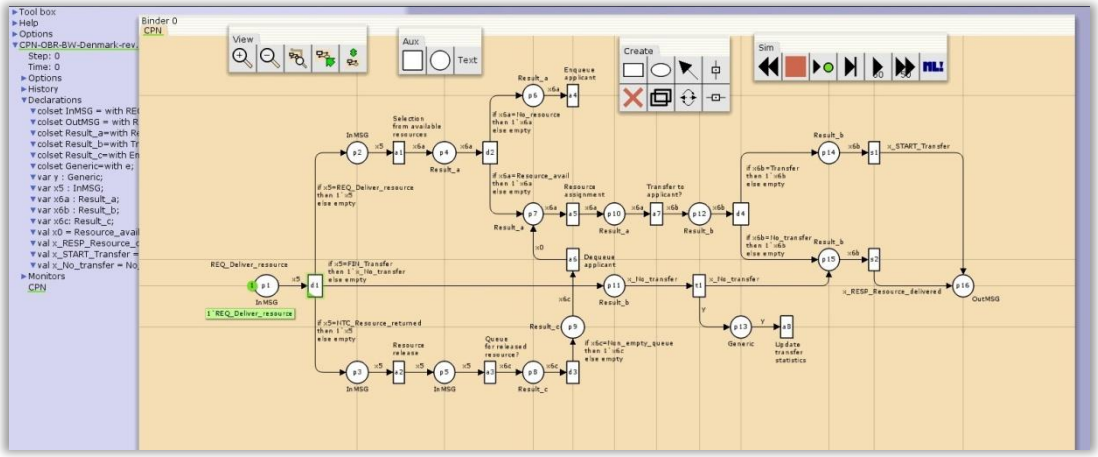
5.1. Editory pro tvorbu ABA-CPN

Aby bylo možné provádět testy na nějaké síti, je nejprve nutné tuto síť vytvořit. Pro tvorbu Petriho sítí existuje mnoho editorů, ale jelikož se v rámci této práce zajímáme jen o podtřídu ABA-CPN, je tento výběr dosti omezen. K vytvoření tohoto typu sítě nám postačí mít k dispozici editor určený pro tvorbu běžných CPN. Takový editor je součástí nástroje CPN Tools.

CPN Tools

CPN Tools je nástroj určený pro editaci, simulaci a pro celou analýzu barvených Petriho sítí. Byl vytvořen na dánské University of Aarhus výzkumnou skupinou, která se zabývá právě zmíněnými barvenými Petriho sítěmi. Samotný program je napsán jazykem JAVA.

Tento nástroj je sice primárně určen pro tvorbu a analýzu CPN, ale jelikož podtřída ABA-CPN z hlediska stavby sítě nic nového proti CPN nepřináší, tak nám pro tvorbu této podtřídy více než postačuje. CPN Tools pracuje se sítěmi prostřednictvím xml souborů s příponou *cpn*. Následující obrázek ukazuje prostředí, jaké CPN Tools nabízí.



Obrázek 8 – Ukázka prostředí CPN Tools

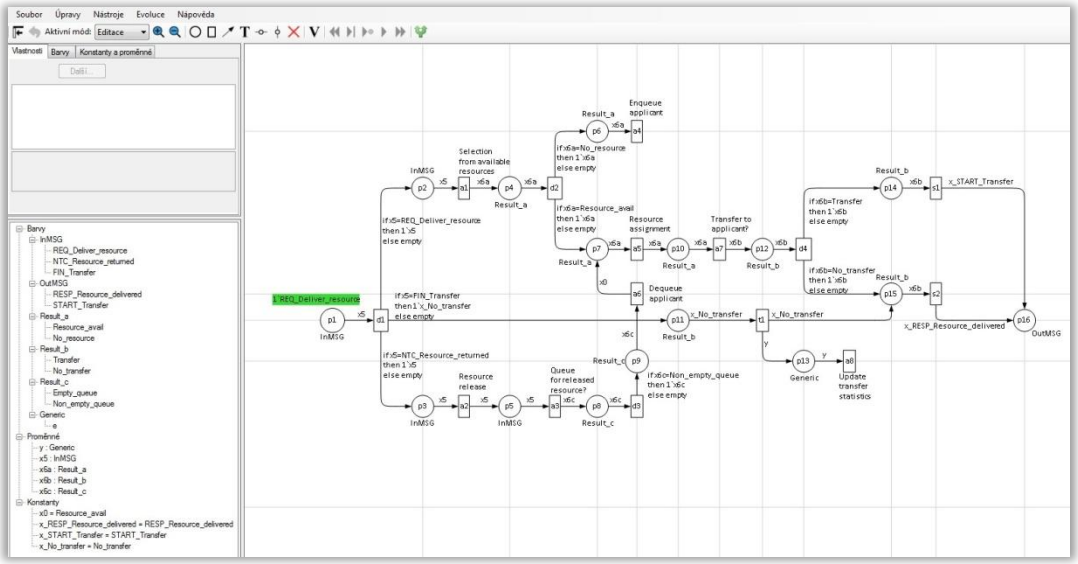
ABA-CPN Editor

Alternativu k CPN Tools nabízí právě editor, který je součástí této práce. Je primárně určen pro tvorbu a editaci ABA-CPN, takže v rámci tohoto typu sítě může posloužit stejně dobře jako CPN Tools. ABA-CPN Editor není psán jazykem JAVA stejně jako předchozí editor, ale jazykem C# v prostředí Microsoft Visual Studio 2010.

Jedním z požadavků na tento editor bylo, že bude pracovat se soubory stejného formátu jako CPN Tools, tzn. s xml soubory stejného formátu, jaký vytváří CPN Tools.

Tento požadavek byl dodržen, a tak je možné soubory vytvořené tímto editorem otevřít i pomocí CPN Tools. Jediný problém, který by mohl nastat, je v názvu ukládaného souboru. Pokud by se zadal název obsahující některé nepovolené znaky (např. znaky s diakritikou), tak CPN Tools tento soubor neotevře.

Pokud tento postup obrátíme, tak soubory vytvořené pomocí CPN Tools nám ABA-CPN Editor nemusí otevřít (nebo načte jen část souboru). Jedinou podmínkou bezproblémového otevření je, že veškerá deklarace (množin barev, proměnných a konstant) vytvářená v CPN Tools je zapsána přímo do bloku *Declarations* (další bloky, vytvořené uvnitř tohoto bloku, se ignorují). Následující obrázek ukazuje prostředí ABA-CPN Editoru se stejnou sítí, která byla otevřena v CPN Tools na předchozím obrázku.



Obrázek 9 – Ukázka prostředí ABA-CPN Editoru

Nedílnou součástí této práce i editoru je interpret podtřídy ABA-CPN. Jelikož se v některých situacích liší řešení přechodů běžné CPN od ABA-CPN (nemluvě o správě instancí, která jde za rámec běžných CPN), tak samotnou evoluci je třeba provádět právě za pomocí tohoto nástroje.

Kromě možného vytvoření sítě a provádění evoluce je další funkcionalitou této aplikace i verifikace právě otevřené sítě. V případě nalezení chyb, které odporují definici ABA-CPN, jsou chybné části sítě graficky zvýrazněny červenou barvou a mimo to se objeví i dialog s výpisem nalezených chyb.

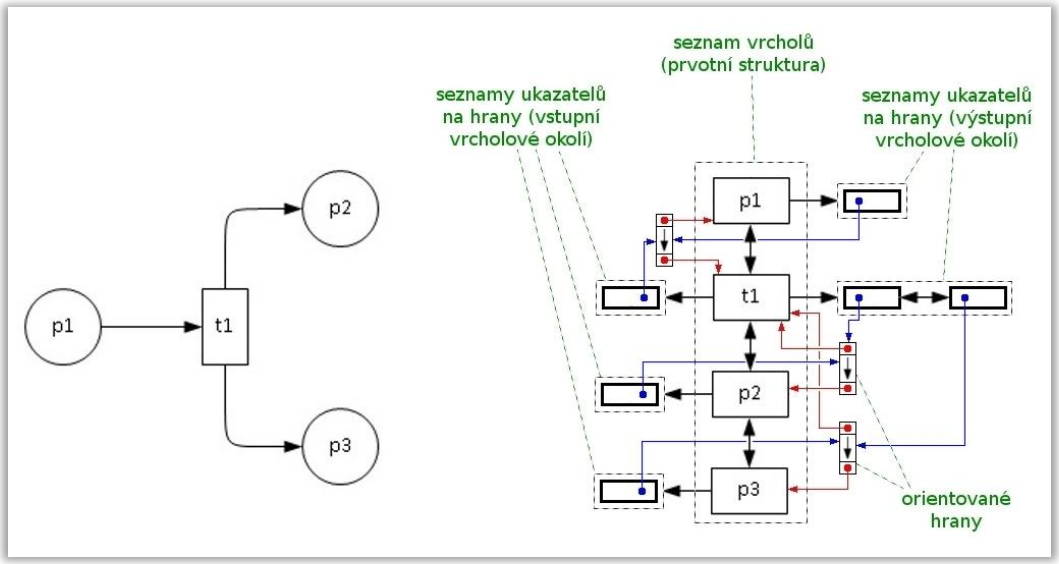
Poslední výraznou součástí aplikace je možnost grafického znázornění vypočítaného stavového prostoru právě otevřené sítě. Na této části aplikace pracoval můj kolega Jiří Engliš, který svou část jistě podrobně popsal ve své diplomové práci *Výpočet stavového prostoru podtřídy barvené Petriho sítě*.

5.2. Datové struktury

V této kapitole budou popsány některé důležité datové struktury využité v rámci ABA-CPN Editoru. Dále zde bude popis některých základních rozhraní a řešení některých problémů.

Datová struktura uchovávající síť

Jako datovou strukturu, která by měla uchovávat ABA-CPN, jsem použil dopředně-zpětnou hvězdu. Prvotní struktura je implementována jako obousměrně zřetězený seznam vrcholů, kde každý z těchto vrcholů má své vlastní obousměrné seznamy (druhotné struktury) ukazatelů na výchozí a příchozí hrany. Příklad takové struktury je na následujícím obrázku.



Obrázek 10 – Ukázka datové struktury pro uchování sítě

Na příkladu z obrázku můžeme vidět tři hrany. Každá z nich obsahuje (mimo jiné) ukazatele na oba vrcholy, které spojuje. Na každou z těchto hran existují vždy dva ukazatele: první je ze seznamu výchozích hran výchozího vrcholu, druhý ze seznamu vstupních hran cílového vrcholu.

Prvotní struktura uchovává vrcholy bez ohledu na to, jestli se jedná o místo či přechod. K její implementaci nebylo nutné použití vyhledávacích struktur (nějaký druh stromu nebo hash tabulku), protože při provádění evoluce stačí získat vstupní vrchol a následně už jsou všechny sousední vrcholy přímo propojeny pomocí hran, a proto již není potřeba nic vyhledávat.

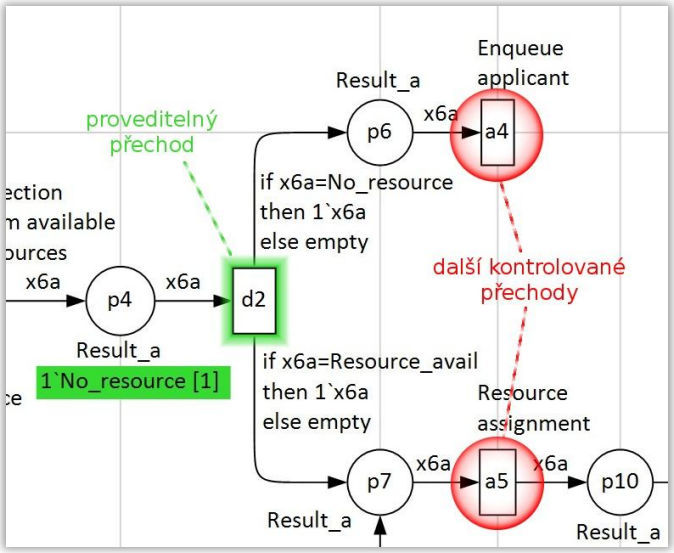
Pro druhotné struktury jsem volil implementaci opět pomocí obousměrného seznamu, protože ve většině případů budou tyto seznamy obsahovat jeden až dva záznamy a za těchto počtů nemá smysl implementovat složitější struktury.

Pro specifické práce se sítí jsou využity další pomocné struktury, které sami o sobě neobsahují žádná důležitá data, ale jen ukazatele do původní struktury sítě.

Struktura spojená s prováděním evoluce ABA-CPN

Pokud síť projde bez chyb verifikací, jež bude popsána v následující kapitole, může se ABA-CPN Editor přepnout do režimu evoluce. To následně znamená vytvoření interpreta ABA-CPN, který si při své tvorbě zjistí vstupní vrchol. Evoluce dále probíhá pomocí provádění (odpalování) proveditelných přechodů. Aby se nemusela po každém provedení přechodu procházet celá síť a hledat všechny přechody, které jsou proveditelné, vytvořil jsem jeden seznam ukazatelů na přechody, jež jsou proveditelné.

Po vytvoření interpreta se zkontroluje přechod, jenž je spojen se vstupním místem. Pokud je proveditelný, zařadí se do seznamu proveditelných přechodů. Z tohoto seznamu je následně proveden jeden náhodný nebo přesně určený přechod a na řadu přichází kontrola dalších přechodů, zda je nezařadit do seznamu proveditelných přechodů. Tato kontrola se netýká celé sítě, ale jen přechodů, které mohly být provedením posledního přechodu ovlivněny. Kontrola směrem vzad od provedeného přechodu nemá význam, protože z míst vychází vždy jen jedna hrana, a proto místa na vstupním vrcholovém okolí ovlivňují pouze jeden přechod (tj. ten, který se provedl). Kontrola se tedy týká přechodů, které jsou spojeny s místy na výstupním vrcholovém okolí prováděného přechodu a samozřejmě se kontroluje i samotný provedený přechod. Ten může být i po provedení znovu proveditelný. Pokud se některý z těchto přechodů stane proveditelným, zařadí se opět do seznamu (pokud tam již není). Evoluce sítě tímto způsobem jednoduše pokračuje, dokud není tento seznam prázdný. Na následujícím obrázku je vidět situace před provedení přechodu a které další přechody se budou po jeho provedení kontrolovat pro možnost zařazení do seznamu proveditelných přechodů.



Obrázek 11 – Ukázka z evoluce

Na obrázku je vidět přechod *d2*, který je v tuto chvíli zařazen do seznamu proveditelných přechodů. Po jeho provedení je potřeba zjistit, jestli se jeho provedením nestal proveditelný nějaký jiný přechod. Nejprve se zjistí, zda je stále proveditelný přechod *d2* a následně se zkontrolují přechody, pro které jsou místa z výstupního vrcholového okolí přechodu d2 (*p6*, *p7*) místy ze

vstupního vrcholového okolí. Tomuto odpovídají přechody *a4* a *a5*. Provedením přechodu *d2*, který je uveden v příkladu, se přesune značka *No_resource* z místa *p4* do *p6*. Přechod *d2* je tímto neproveditelný (odebere se ze seznamu) a po kontrole přechodů *a4* a *a5* se zjistí, že je proveditelný pouze přechod *a4*, tj. ukazatel na tento přechod se vloží do seznamu proveditelných přechodů.

Práce s grafickým znázorněním sítě

Na grafické znázornění jsem měl několik kritérií. První kritériem bylo vykreslovat objekty v určitém pořadí (např. nejprve všechna vodítka, potom hrany, …). Dále by bylo výhodné všechny objekty, které by se měli vykreslit, procházet co nejjednodušším a nejrychlejším způsobem. Posledním kritériem bylo rozlišování objektů, které jsou vykresleny částečně nebo dokonce úplně přes sebe, na než bylo kliknuto myší.

Pro všechny objekty, které by se měly vykreslovat, jsem vytvořil jednotné rozhraní *IElementObrazu*, jež musely implementovat.

Obrázek 12 – IElementObrazu

Toto rozhraní obsahuje mimo jiné i nutnost implementace standardního rozhraní *IComparable*, které obsahuje metodu, s jejíž implementací lze následně řadit jednotlivé elementy obrazu.

Rozhraní *IElementObrazu* dále obsahuje vlastnost pouze pro čtení, která udává, v jakém pořadí se má daný element vykreslit. Součástí rozhraní jsou i tři metody *ObsahujeBod*, *Vykresli* a *Zvyrazni*.

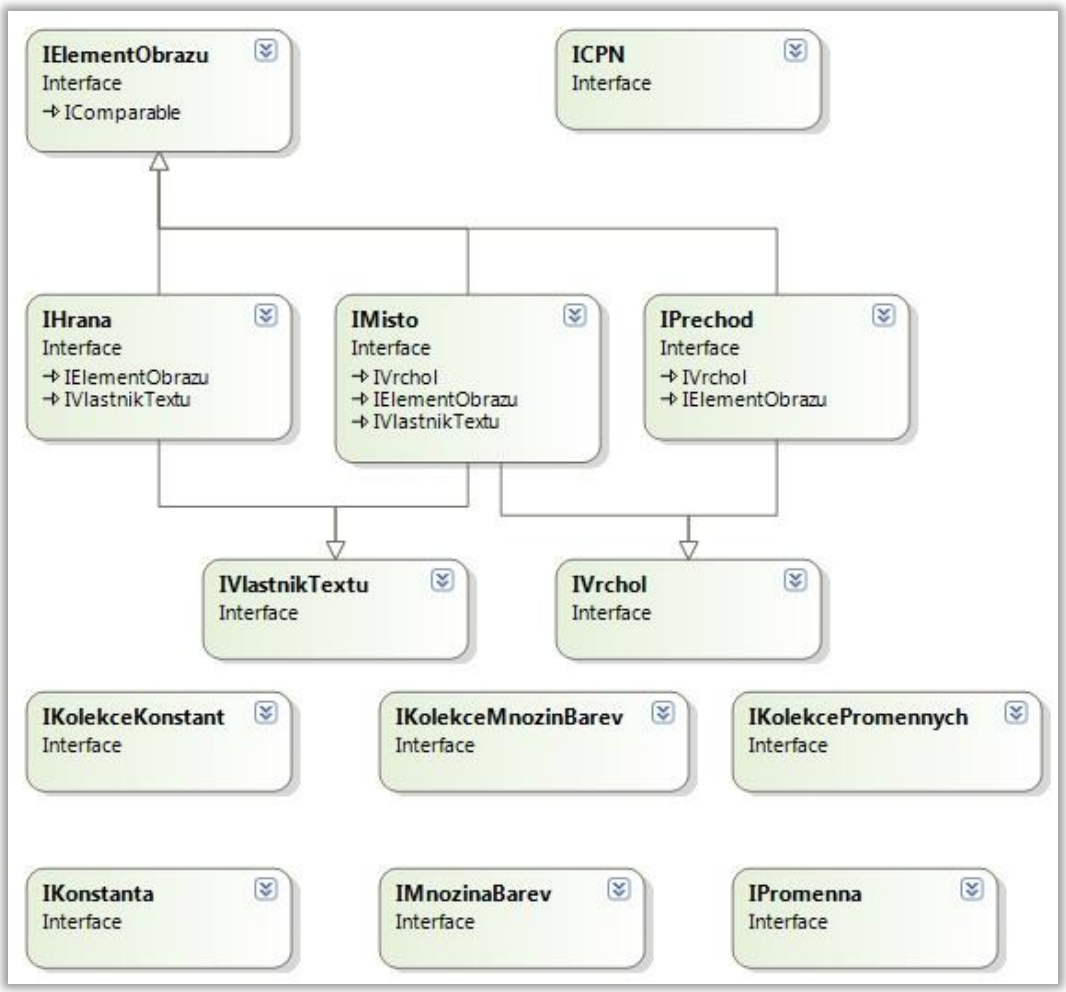
První metodou (*ObsahujeBod*) s parametrem typu *Point* si lze ověřit, zda nějakou částí plochy dotazovaný element obrazu leží na zadaném bodu. Druhá metoda (*Vykresli*) slouží logicky k vykreslení elementu. Poslední metodou (*Zvyrazni*) vykreslíme podbarvení barvou uvedenou v parametru metody.

Aby bylo možné všechny elementy obrazu jednoduše procházet a vykreslovat, byla pro tyto účely vytvořena pomocná datová struktura utříděný seznam. Třídí se za pomoci metody *CompareTo*, která je součástí výše zmíněného rozhraní *IComparable* a dokáže určit, zda je objekt v parametru této metody z pohledu pořadí vykreslení dříve, později, nebo na stejné úrovni. Seznam se udržuje v utříděném stavu tak, že se elementy již vkládají v utříděném pořadí. Tento seznam tedy obsahuje ukazatele do struktury sítě a dále také na všechny ostatní elementy, které se vykreslují (vodítka, textové poznámky). Pro vykreslení už dále není třeba složitého procházení několika struktur, ale stačí projít tento seznam, který je navíc utříděn tak, aby se elementy vykreslil ve správném pořadí. Stejně tak při kliknutí myši do obrazu stačí projít znovu jen tento seznam (tentokrát v opačném pořadí) a element, který jako první oznámí, že se na daném bodě vyskytuje, je právě ten, jenž je ze všech elementů v tomto bodě „nejvýše“ (vykreslen jako poslední).

Je nutné dodat, že při vkládání či odebírání nějakého objektu ze struktury sítě je třeba provést stejné změny i s tímto utříděným seznamem, aby nenastala situace, kdy odebereme např. přechod a on se nám i přesto stále vykresluje, jako kdyby byl ještě součástí sítě, nebo naopak nějaký přechod přidáme a on se nevykreslí.

Rozhraní aplikace

Při tvorbě této aplikace bylo nutné stanovit základní definice používaných objektů. Důvod k tomuto opatření byl takový, že jsme s kolegou Englišem pracovali více méně odděleně a každý si potřebné třídy definoval sám. Aby závěrečné propojování našich aplikací neprovázeli větší problémy, rozhodli jsme se pro každý takový klíčový objekt definovat specielní rozhraní, které budeme v aplikaci používat. Díky tomuto opatření se propojování aplikací obešlo bez větších problému a také se tím výsledná aplikace stala nezávislou na použitých strukturách. Použitá rozhraní jsou na následujícím obrázku. Pro jednoduchost jsou zobrazeny jen jejich názvy.



Obrázek 13 – Použitá rozhraní

Na obrázku můžeme vidět celkem 13 různých rozhraní, jejichž význam zde bude vysvětlen. Konkrétní přehled obsahů všech rozhraní nemá pro tento text větší smysl, a tak tu tyto obsahy nejsou uvedeny. U některých rozhraní jsou vypsány jen namátkou některé typy vlastností nebo metod, aby bylo zřejmé, co přibližně obsahují.

- **ICPN** – Rozhraní, které definuje základní metody pro práci se sítí. Patří sem metody typu *vlož*, *najdi* a *odeber* a dále také vlastnosti umožňující zjistit počty jednotlivých typů vložených elementů.

- **IElementObrazu** – Toto rozhraní zaštiťuje veškeré vykreslované objekty a poskytuje metody pro prácí s jejich grafickou reprezentací. Podrobnější popis byl uveden v části *Práce s grafickým znázorněním sítě*.

- **IVrchol** – Rozhraní, které obsahuje obecné metody a vlastnosti, jež jsou společné pro oba typy vrcholů (místo a přechod). Obsahuje souřadnice, ukazatele na seznamy vstupních a výstupních hran apod.

- **IVlastnikTextu** – Jedná se o rozhraní které má pouze jednu vlastnost vracející referenční bod. Toto rozhraní je implementováno jen objekty, ke kterým se váže nějaký „plovoucí“ text, s nímž se může hýbat. Takové objekty jsou místa, ke kterým se může vázat výpis značek, a hrany, vlastnící nějaký výraz (proměnnou, konstantu, if …). Referenční bod se využije při posunu objektu, aby se stejným způsobem mohl posunout i vlastněný text.

- **IHrana** – Toto rozhraní slouží pro definici všech potřebných metod a vlastností, které jsou od hrany aplikací očekávány. Jsou tam uvedeny např. ukazatele na výchozí a cílový vrchol, informace týkající se grafického znázornění (šířka, seznam bodů pro zalamování hrany, barva čáry, …), informace o výrazu na popisu hrany, apod. Součástí tohoto rozhraní je i implementace rozhraní *IVlastnikTextu* a *IElementObrazu*.

- **IMisto** – Podobně jako *IHrana* je i toto rozhraní definicí vlastností a metod, které jsou v aplikaci požadovány po objektu reprezentujícím místo. Rozhraní definuje opět vlastnosti týkající se grafického znázornění, informace o obsažených značkách, druh místa atd. Implementací tohoto rozhraní je třeba implementovat i další tři rozhraní, která s tímto souvisí. Jsou to *IElementObrazu*, *IVlastnikTextu* a *IVrchol*.

- **IPrechod** – Je to poslední rozhraní popisující jednu ze základních součástí sítě a to přechod. I toto rozhraní souvisí s implementací jiných rozhraní: *IElementObrazu* a *IVrchol*.

- **IKolekceKonstant, IKolekcePromennych, IKolekceMnozinBarev** – Tato tři rozhraní definují vlastnosti a metody kolekcí, kam se ukládají konstanty, proměnné a množiny barev.

- **IKonstanta, IPromenna, IMnozinaBarev** – Jak je již z názvu patrné, účel těchto rozhraní je definice konstanty, proměnné a množiny barev, které se ukládají do kolekcí definovaných předchozími rozhraními. Značky (barvy) jsou ukládány do množin barev v podobě řetězce znaků (*string*).

5.3. Verifikace sítě

Dalším požadavkem na aplikaci byla možnost provést verifikaci sítě. Následující podkapitola je věnována právě prováděným testům sítě. Tyto testy se dají rozdělit do pěti částí:

1. kontrola názvů v deklaraci,
2. kontrola míst,
3. kontrola přechodů,
4. kontrola hran a
5. kontrola acykličnosti.

Kontrola názvů v deklaraci

Tato kontrola postupně prochází množiny barev i s jejich obsahy, proměnné a konstanty a za chybový stav považuje následující:

- duplicita názvů v deklaraci,
- množina barev je prázdná,
- deklarace obsahuje klíčové slovo *empty*,
- deklarace obsahuje klíčové slovo *if*,
- deklarace obsahuje klíčové slovo *then* a
- deklarace obsahuje klíčové slovo *else*.

Kontrola míst

Tento druh kontroly prochází pochopitelně všemi místy a zjišťuje, zda je vše v pořádku. Za chybu je zde považováno, pokud:

- místo nemá přidělenou množinu barev,
- místo obsahuje neznámou (nedeklarovanou) množinu barev,
- z místa nevychází právě jedna hrana (nepočítáno pro výstupní místo),
- z výstupního místa vycházejí hrany,
- do místa neústí žádná hrana (nepočítáno pro vstupní místo),
- do vstupního místa ústí hrany,
- vstupní místo neobsahuje žádnou inicializační značku,
- vstupní místo obsahuje více než jednu inicializační značku,
- vstupní místo obsahuje inicializační značku, která nespadá do množiny barev tohoto místa,
- síť neobsahuje žádné vstupní místo,

- síť obsahuje více než jedno vstupní místo,
- síť neobsahuje žádné výstupní místo,
- síť obsahuje více než jedno výstupní místo.

Kontrola přechodů

Podobně jako předchozí kontrola tato projde všemi přechody a zjišťuje následující chyby:

- z rozhodovacího přechodu nevycházejí jen rozhodovací hrany,
- do přechodu neústí žádná hrana,
- do asistenčního přechodu ústí více než jedna hrana,
- do rozhodovacího přechodu ústí více než jedna hrana,
- do odesílacího přechodu ústí více než jedna hrana,
- z rozhodovacího přechodu nevychází žádná hrana,
- z odesílacího přechodu nevychází žádná hrana.

Kontrola hran

Tato kontrola opět projde všemi hranami a hledá možné chyby. Jelikož aplikace přímo rozlišuje tři druhy hran, tak je tato kontrola také rozložena na tři možné varianty (podle toho, která hrana se má právě kontrolovat).

1. Kontrola elementární hrany odhaluje následující chyby:
   - elementární hrana nedisponuje žádným hranovým výrazem,
   - výraz elementární hrany nabývá neznámé hodnoty (neshoduje se s žádnou deklarovanou proměnnou),
   - výraz elementární hrany obsahuje proměnnou, která nespadá pod množinu barev incidentního místa.
2. Kontrola konstantní hrany považuje za chybu:
   - konstantní hrana nedisponuje žádným hranovým výrazem,
   - výraz konstantní hrany nabývá neznámé hodnoty (neshoduje se s žádnou deklarovanou konstantou ani barvou),
   - výraz konstantní hrany obsahuje konstantu, která nespadá pod množinu barev incidentního místa,
   - výraz konstantní hrany obsahuje barvu, která nespadá pod množinu barev incidentního místa.
3. Kontrola rozhodovací hrany zjišťuje tyto chyby:
   - rozhodovací hrana nevychází z rozhodovacího přechodu,

- rozhodovací hrana nedisponuje žádným hranovým výrazem,
- výraz rozhodovací hrany neobsahuje proměnnou ze vstupu incidentního přechodu,
- výraz rozhodovací hrany obsahuje nesplnitelnou podmínku (proměnná v podmínce nemůže nikdy obsahovat hodnotu podmínky, tj. hodnota nespadá pod množinu barev proměnné),
- výraz rozhodovací hrany obsahuje pro *then* i *else* hodnoty *empty*,
- výraz rozhodovací hrany nabývá pro *then* proměnné, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *else* proměnné, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *then* konstanty, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *else* konstanty, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *then* barvy, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *else* barvy, která nespadá pod množinu barev incidentního místa,
- výraz rozhodovací hrany nabývá pro *then* neznámé hodnoty,
- výraz rozhodovací hrany nabývá pro *else* neznámé hodnoty.

Kontrola acykličnosti

Touto kontrolou prověřujeme, zda síť splňuje požadavek acykličnosti a zákaz vlastních cyklů. Aby bylo možné tuto kontrolu provést, musí se tentokrát pomocí specielního algoritmu prozkoumat celá síť. Jedná se vlastně o upravenou prohlídku do hloubky, která není nasazena na stromovou strukturu, ale na síť, kterou můžeme díky znalosti vstupního místa považovat za strom s kořenem umístěným právě ve vstupním místě. Jelikož nelze při verifikaci zaručit, aby síť obsahovala právě jedno vstupní místa, tak je v algoritmu pro tato místa vytvořen celý seznam. I přes možnost existence více vstupních míst pracuje algoritmus stále správně.

Algoritmus během chodu přiděluje vrcholům vždy jedno ze tří označení:

- *fresh*, které značí, že vrchol nebyl algoritmem dosud navštíven,
- *open*, což znamená, že vrchol již byl algoritmem navštíven a dosud je součástí právě prověřované trasy a
- *closed* je přidělován vrcholům, které už byly algoritmem prověřeny a už nadále nejsou součástí právě prověřované trasy.

Algoritmus můžeme popsat následujícími kroky:

Krok 1. Vytvoření a naplnění seznamu vstupních míst (*svm)*.

Krok 2. Vytvoření zásobníku.

Krok 3. Inicializace všech vrcholů sítě na *fresh*.

Krok 4. Pokud je *svm* prázdný, tak následuje krok 13, jinak se z *svm* odebere libovolné místo, přiřadí se mu označení *open* a vloží se do zásobníku.

Krok 5. Pokud je zásobník prázdný, tak následuje krok 4, jinak budeme v dalších krocích pracovat s vrcholem *v*, který se nachází na vrcholu zásobníku.

Krok 6. Za existence minimálně jedné hrany, která vychází z vrcholu *v*, přejdeme na krok 7, jinak následuje krok 8.

Krok 7. Vložíme do zásobníku první vrchol z výchozího vrcholového okolí vrcholu *v*, který bude mít označení *fresh*. Pokud by žádný takový vrchol neexistoval, přejdeme na krok 8, jinak na krok 5.

Krok 8. Pokud zásobník není prázdný, budeme v dalších krocích označovat vrchol nacházející se na vrcholu zásobníku jako *v2*, jinak následuje krok 5.

Krok 9. Pokud stále existuje vrchol z výstupního vrcholového okolí vrcholu *v2*, který má označení *fresh*, přejdeme na krok 5, jinak následuje krok 10.

Krok 10. Vrchol *v2* označíme *closed*.

Krok 11. Prohledáme všechny vrcholy z výchozího vrcholového okolí vrcholu *v2*, a pokud bude mít některý z nich označení *open*, je to známka toho, že přes tento vrchol můžeme znovu vstoupit do aktuálně prohledávané trasy, a tudíž jsme našli cyklus.

Krok 12. Odebereme ze zásobníku vrchol *v2* a pokračujeme krokem 8.

Krok 13. Konec algoritmu.

Tento algoritmus tedy začíná vždy ve vstupním místě a následně „vytváří“ trasu danou prohlídkou do hloubky. Trasa je tedy množina vrcholů mezi aktuálním vrcholem a vstupním místem. Vrcholy, které jsou součástí této trasy, mají vždy označení *open* a ostatní *fresh* nebo *closed*. Pokud tedy trasa pomocí prohlídky do hloubky narazí na vrchol s označením *open*, znamená to, že se vrací na jeden z vrcholů této trasy, tj. existuje tu cyklus, jenž je dán vrcholy, které můžeme zjistit, pokud budeme postupovat zpět po trase, dokud nenarazíme opět na stejný vrchol.

6. Testování

Nedílnou součástí vývoje aplikace je i testování, na které je zaměřena tato kapitola. Bude zde testován hlavně interpret ABA-CPN, které musí správně vyhodnotit situaci kolem přechodu tak, aby se přechod provedl správně. Budou zde testovány i extrémní situace, které v praxi téměř nemohou nastat, pro které jsou vytvořeny speciální sítě. Dále zde bude také prověřen generátor náhodných čísel, aby bylo zřejmé jeho rozdělení pravděpodobnosti při potřebě náhodně vybrat jednu z možností (např. výběr jedné z několika značek nebo náhodný výběr jednoho proveditelného přechodu).

6.1. Přenášení instance

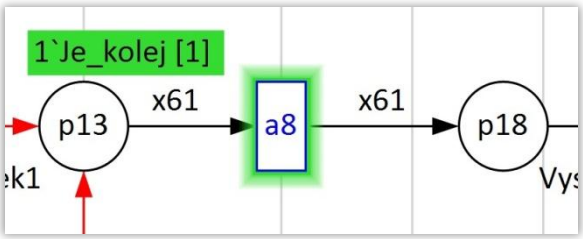
Tato podkapitola je určena pro ukázku testování interpreta ABA-CPN. Budou zde ukázány některé běžné případy provádění přechodů, kde bude zapotřebí nejen správně přenést značky, ale hlavně také správně vyřešit přenášení jejich instancí (v příkladech budou čísla instancí zapsána vedle značek v hranatých závorkách).

V příkladech zanedbáme možnost, že na výstupu přechodu nebude žádné místo. Takový přechod pouze odebere značku ze vstupního místa a nikam už ji nepřenese, tudíž o přenášení instancí nemůže být v tomto případu řeč.

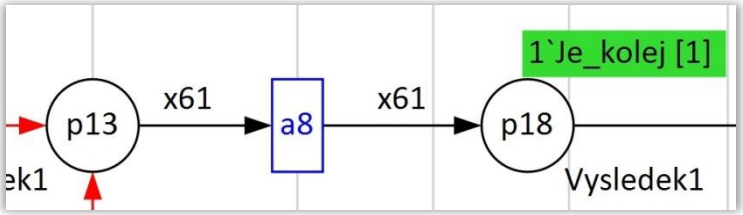
Už i jeden z nejjednodušších případů přechodu, tj. jedno místo na vstupu a jedno na výstupu, má několik různých variant, viz následující příklady.

Přechody příklad 1

Nejjednodušší variantou je, pokud v rámci provádění přechodu existuje pouze jedna proměnná. Touto proměnnou vezmeme ze vstupního místa značku a vložíme ji i s její instancí do výstupního místa.



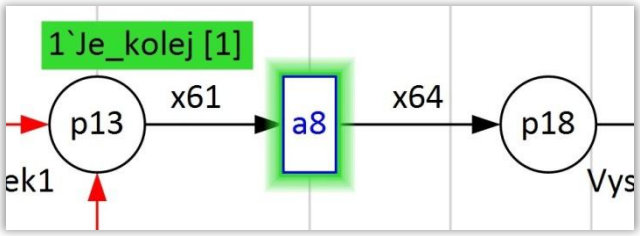
Obrázek 14 – Př. 1 před provedením



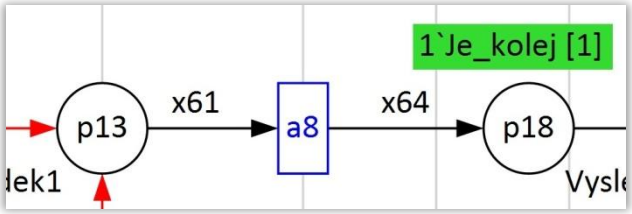
Obrázek 15 – Př. 1 po provedení

Přechody příklad 2

Další možností tohoto typu přechodu je varianta, kdy na každé straně přechodu bude jiná proměnná (definovaná na stejné množině barev) a obě tyto proměnné budou mít stejný prefix (první znak názvu). Vyřešení tohoto přechodu bude stejné jako v předchozím případě, protože proměnná na výstupu přechodu převezme od proměnné na vstupu spolu s instancí i značku. Značka se přenese spolu s instancí právě proto, že jsou obě proměnné definované na stejné množině barev.



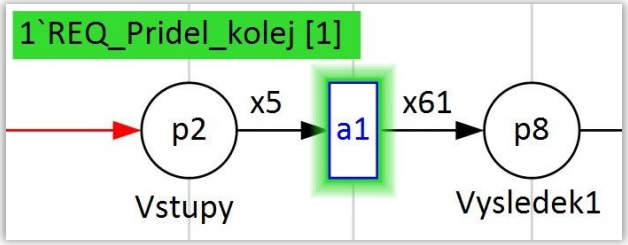
Obrázek 16 – Př. 2 před provedením



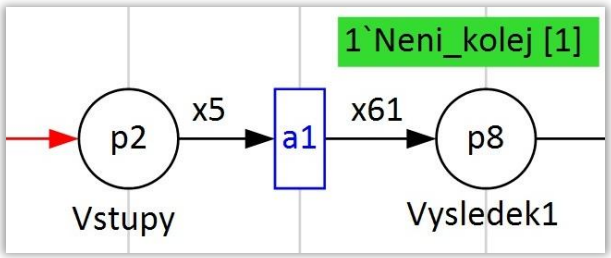
Obrázek 17 – Př. 2 po provedení

Přechody příklad 3

Další změnou proti předchozímu příkladu bude případ, kdy obě proměnné budou mít stejný prefix, ale už budou definované na různých množinách barev. Bude to mít za následek přenesení stejné instance, ale značka se náhodně vygeneruje z množiny barev cílového místa.



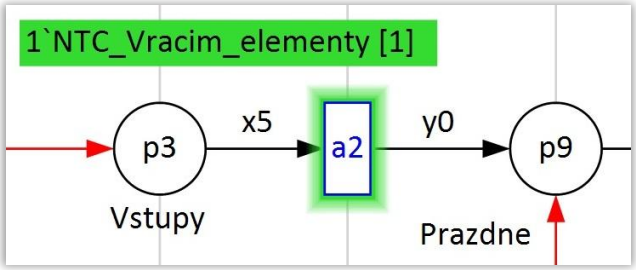
Obrázek 18 – Př. 3 před provedením



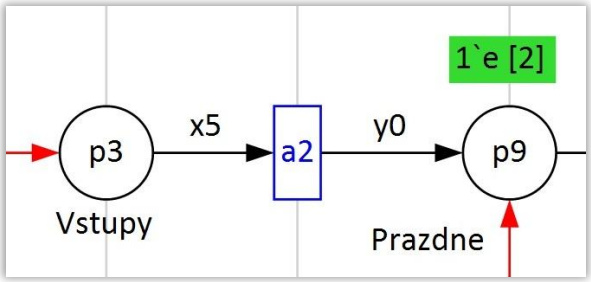
Obrázek 19 – Př. 3 po provedení

Přechody příklad 4

Poslední dvě možnosti tohoto typu přechodu se budou řešit stejně. Opět budeme mít dvě různé proměnné, které budou tentokrát mít různý prefix. V prvním případě budou obě proměnné definované na stejné množině barev a v druhém případě budou mít různé množiny barev. Jelikož se liší jejich prefixy, nepřenese se instance, a tak i v případě stejných množin barev bude generována náhodná značka z množiny barev cílového místa.



Obrázek 20 – Př. 4 před provedením



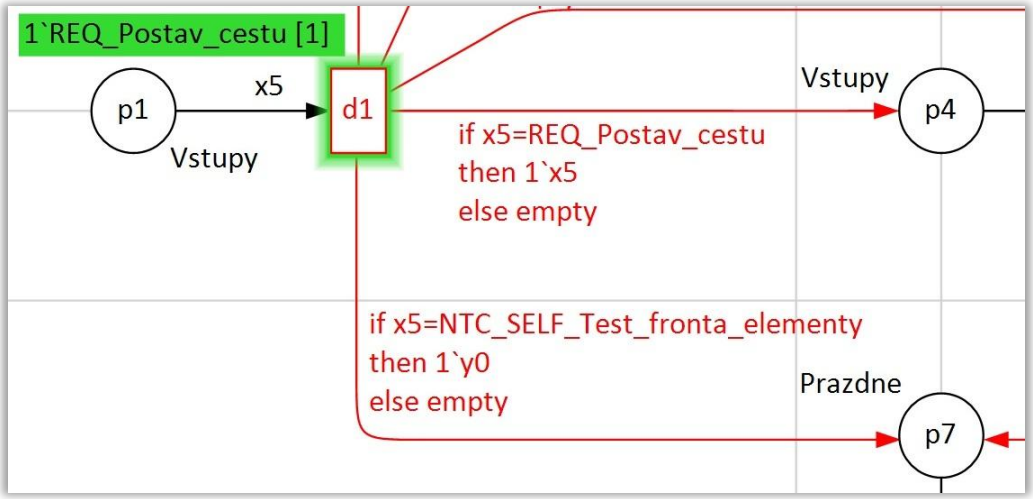
Obrázek 21 – Př. 4 po provedení

Stejným způsobem se provádějí přechody i s konstantami, jen je třeba poukázat na skutečnost, že v případě lokalizace konstanty na výstupu přechodu, by se nikdy žádná značka negenerovala (je dána konstantou) a v případě umístění konstanty na vstupu by se odebrala vždy jen značka odpovídající konstantě. Přenášení instancí díky prefixu se vztahuje i na konstanty.

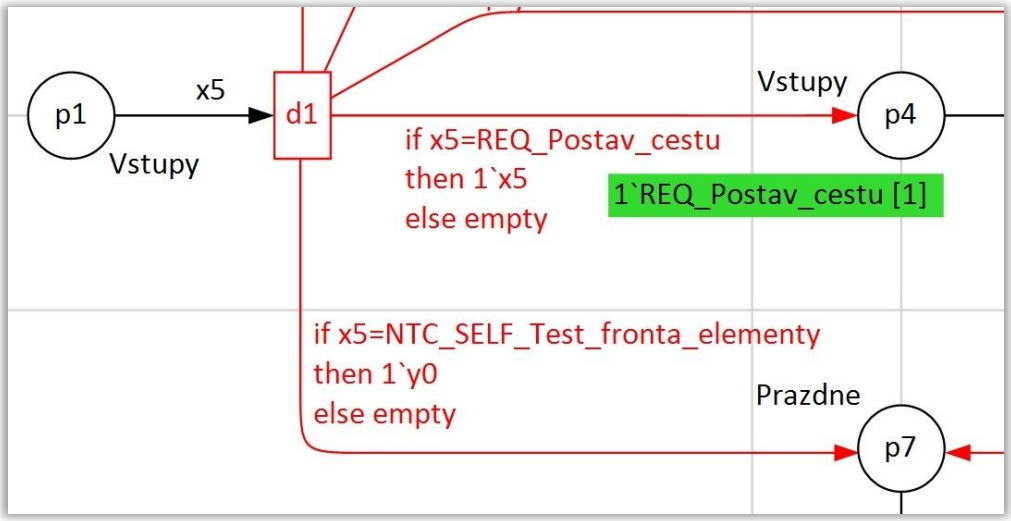
Další příklady předpokládají, že se provádí jiný než standardní přechod, tj. na vstupu bude vždy jen jedna proměnná (nebo konstanta) a na výstupu *n* proměnných (nebo konstant).

Přechody příklad 5

Dalším příkladem bude rozhodovací přechod, který má na vstupu proměnnou (konstanta by neměla smysl, nebylo by co rozhodovat) a na výstupu několik možných výrazů rozhodovacích hran. Nás budou v následujících dvou příkladech zajímat pouze dvě výstupní hrany. V tomto příkladu bude „horní“ výraz vyhodnocen kladně, a tak se stejně jako v příkladu 1 přenese instance spolu se značkou stejnou proměnnou. „Spodní“ výraz se vyhodnotí záporně, a tak nastane možnost *empty*, tj. hrana nic nepřenese.



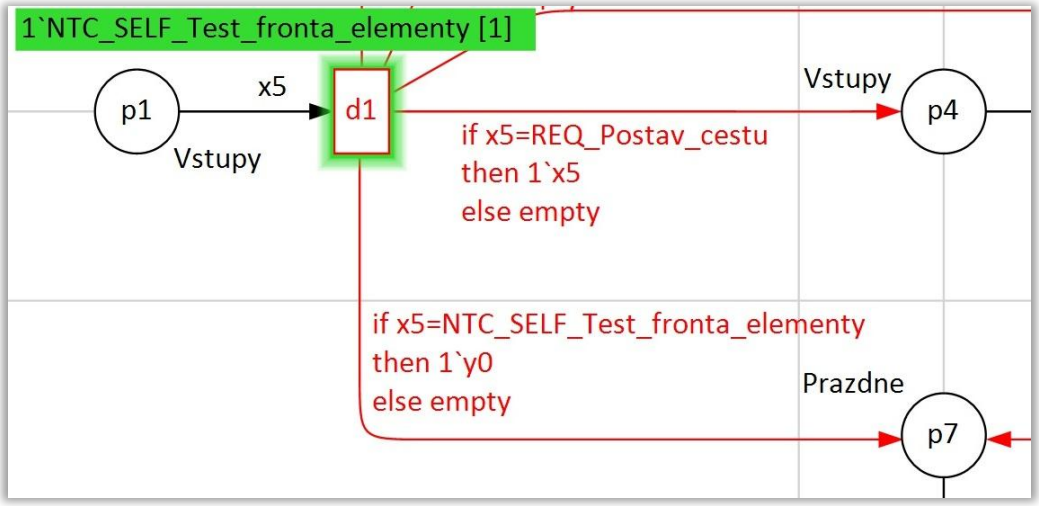
Obrázek 22 – Př. 5 před provedením



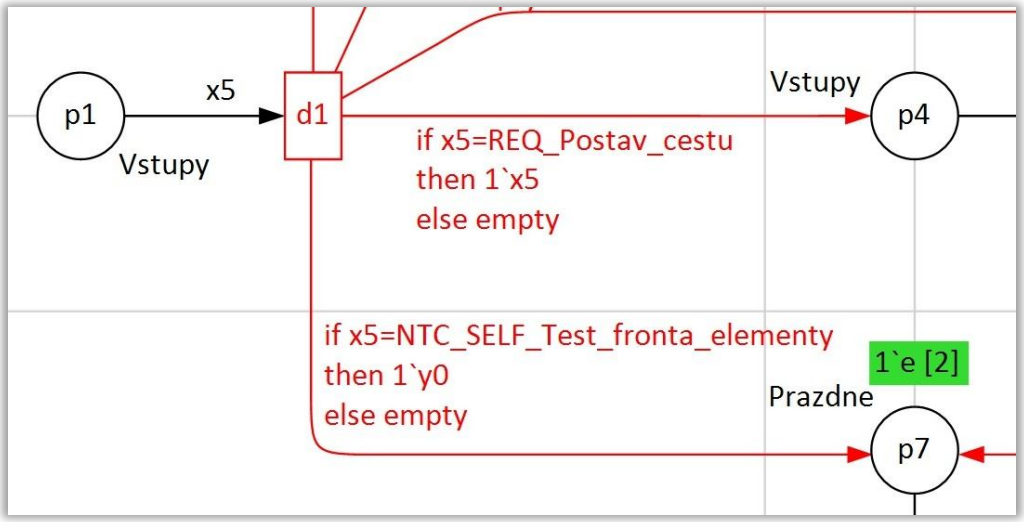
Obrázek 23 – Př. 5 po provedení

Přechody příklad 6

Jedná se o absolutně stejný přechod, jen na vstupním místě bude jiná značka. Tato značka bude znamenat pro „horní“ výraz výstupní hrany možnost *empty* a „spodní“ výraz se vyhodnotí kladně, tj. na vstupu *x5* a na výstupu *y0* a řešení je stejné jako v příkladu 4.



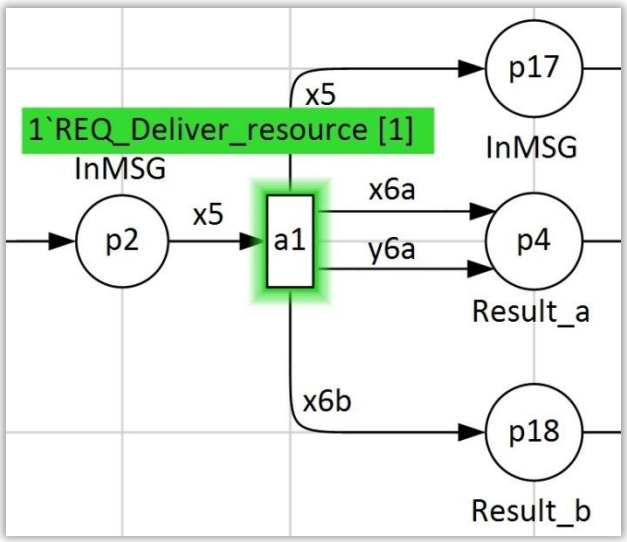
Obrázek 24 – Př. 6 před provedením



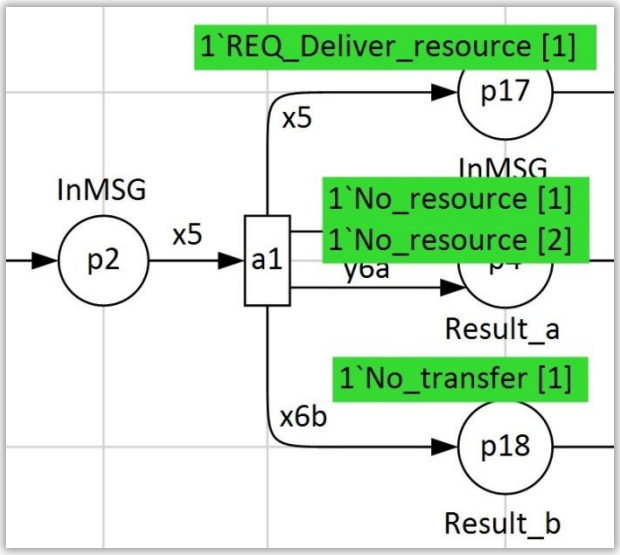
Obrázek 25 – Př. 6 po provedení

Přechody příklad 7

Poslední příklad této podkapitoly je přechod, který má na vstupu proměnnou a na výstupu hned několik proměnných.



Obrázek 26 – Př. 7 před provedením



Obrázek 27 – Př. 7 po provedení

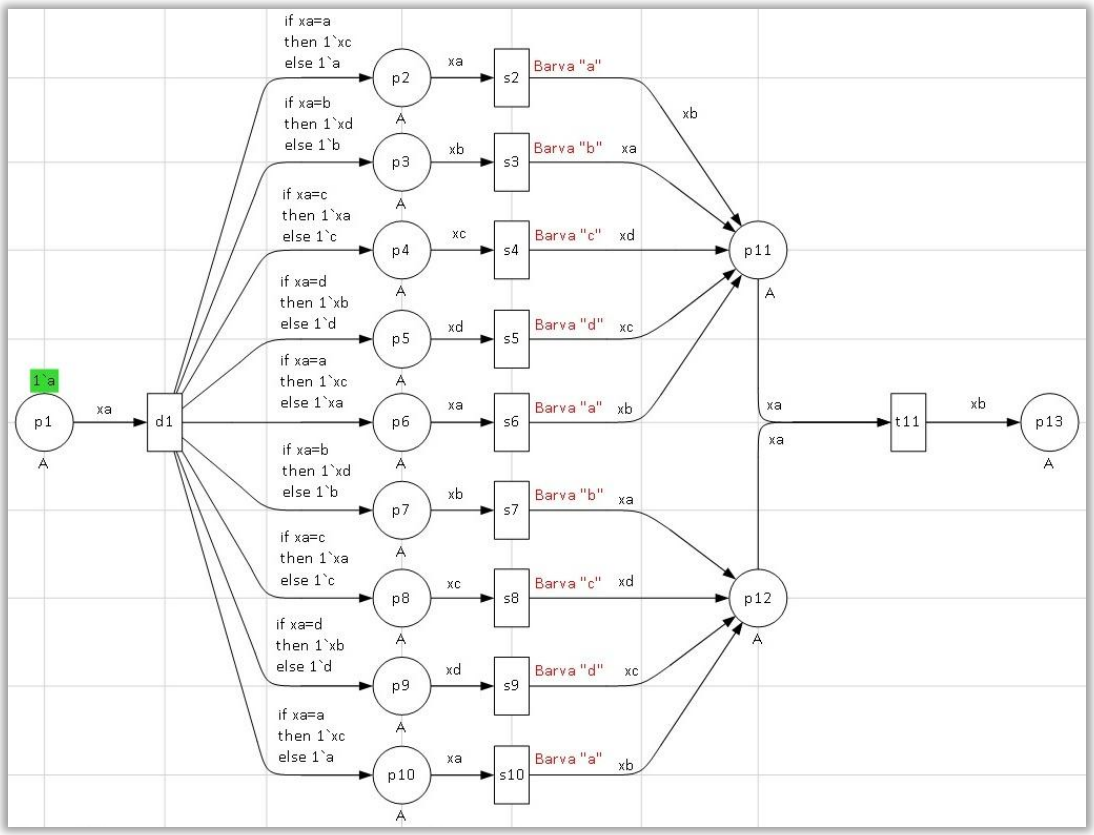
Na tomto posledním příkladě můžeme vidět hned několik předešlých situací najednou. Do místa *p17* se přenesla totožná instance (nebo její kopie) pomocí stejné proměnné jako byla na vstupu (*x5*). Do místa *p4* vedou rovnou dvě hrany s proměnnými *x6a* a *y6a*. Obě tyto proměnné jsou definované na jiné množině barev než vstupní *x5*, tj. jejich hodnota se bude náhodně generovat (v příkladu si obě proměnné shodou okolností vygenerovaly stejnou značku *No_resource*). Díky prefixům se do místa *p4* přenesla pomocí *x6a* stejná instance a pomocí *y6a* se vytvořila nová instance značky. Poslední místo *p18* má opět jinou množinu barev než *p2*, takže se bude značka opět generovat (vygenerovala se značka *No_transfer*) a instance se díky stejným prefixům *x6b* a *x5* přenese.

6.2. Konstrukce extrémních sítí

Pro testování některých velmi ojedinělých situací, které samozřejmě musí interpret zvládnout, se museli vytvořit sítě, které jsou vyloženě určeny pro testování těchto situací. V běžné praxi by tyto sítě neměly žádný význam.

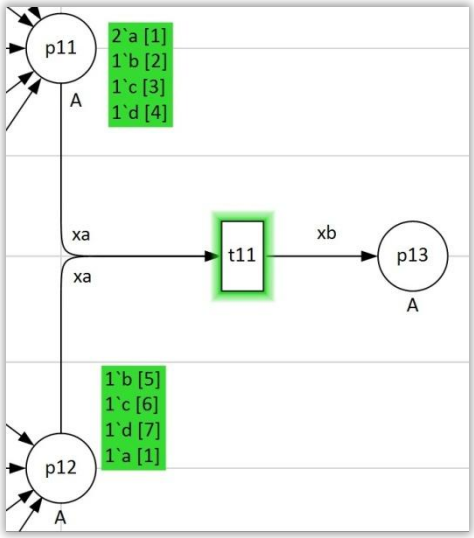
Extrémní síť 1

První síť vypadá následovně (je i součástí přílohy na CD v podobě souboru *moje_sit.cpn*).



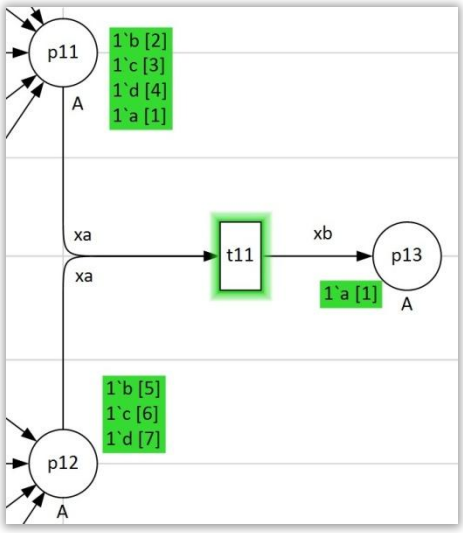
Obrázek 28 – Extrémní síť 1

Celá síť je zkonstruována tak, aby se navodila situace, kdy na vrcholech *p11* a *p12* budou různé značky a instance. Přechod *t11* je potom v situaci, kdy pomocí jedné proměnné na vstupu (tato proměnná je obsažena v obou vstupních hranách) musí umět správně odebrat značky a instance a pomocí jiné proměnné na výstupu vyřešit vložení značek na výstupní místo. Tuto situaci ukazuje následující obrázek.



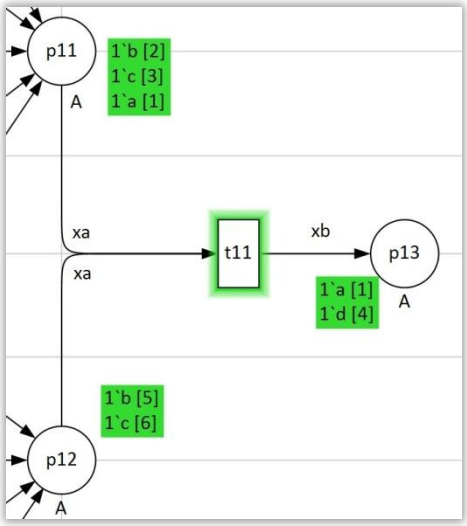
Obrázek 29 – Extrém př. 1 část 1

V tuto chvíli musíme pomocí proměnné *xa* odebrat značky ze vstupních míst. Jelikož se ale jedná o jedinou proměnnou, která v rámci provádění přechodu může přenést pouze jedinou značku, tak se musí vybrat jedna značka, obsažená v obou vstupních místech (bez ohledu na instance) a z obou těchto míst je ji třeba odebrat. Následně je pomocí proměnné *xb*, která má stejný prefix jako *xa* a je ze stejné množiny barev, přenesena i s instancí do výstupního místa *p13*. Na dalším obrázku je vidět, že se náhodně vybrala značka *a*, odebrala se ze vstupních míst a následně se spolu s instancí přenesla do místa *p13*.



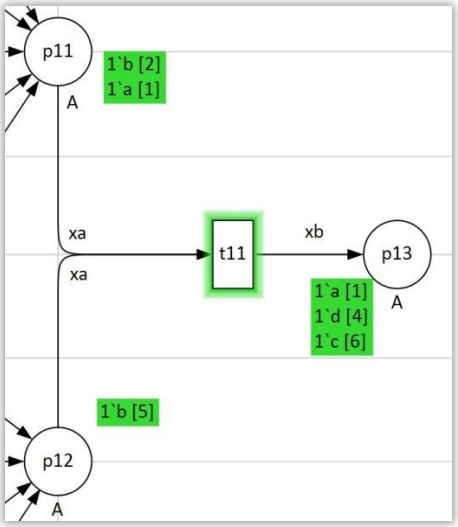
Obrázek 30 – Extrém př. 1 část 2

Následující obrázek řeší aktuální situaci stejný způsobem, ale můžeme si všimnout, že přenášená značka *d* má v obou vstupních místech různé instance. Řešení spočívá v náhodném výběru jedné z instancí, které se odeberou (v příkladu se přenáší instance z místa *p11*).

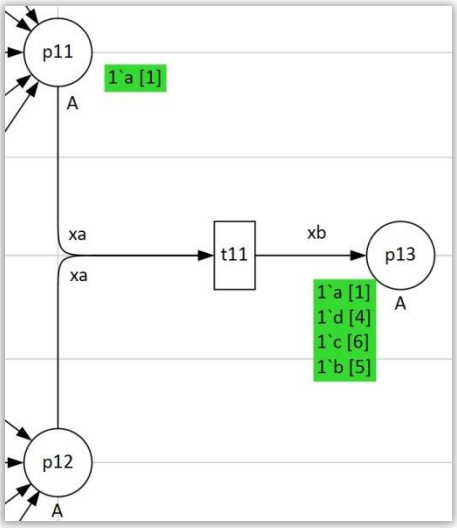


Obrázek 31 – Extrém př. 1 část 3

Poslední dva obrázky jen ukazují dořešení situace. U prvního obrázku se přenesla značka *c* s instancí z místa *p12* a na druhém obrázku značka *b* s instancí opět z místa *p12*. Můžeme si povšimnout, že na posledním obrázku je stále v místě p11 značka, ale přechod již není proveditelní. Důvod je takový, že pro proveditelnost přechodu je nutné, aby na všech vstupních přechodech byla alespoň jedna značka.



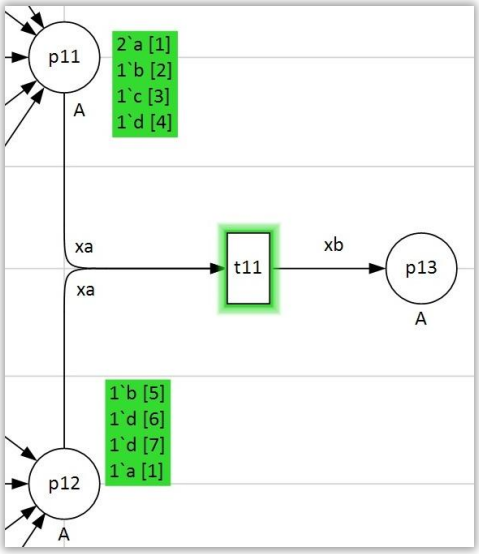
Obrázek 33 – Extrém př. 1 část 4



Obrázek 32 – Extrém př. 1 část 5

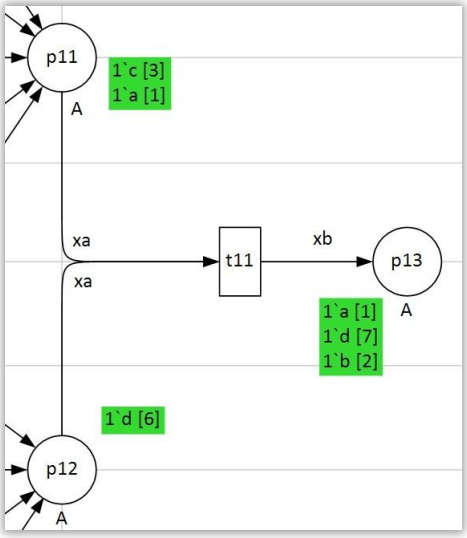
Extrémní síť 2

Druhou síť můžeme opět najít v příloze na CD pod názvem *moje_sit2.cpn*. Jde prakticky o stejnou síť, kde je jen u jedné rozhodovací hrany pozměněn výraz tak, aby se dalo dosáhnout opět na přechodu *t11* specifického stavu. Následující obrázek ukazuje opět situaci, kdy se dosud nepřenesla ani jedna značka.



Obrázek 34 – Extrém př. 2 část 1

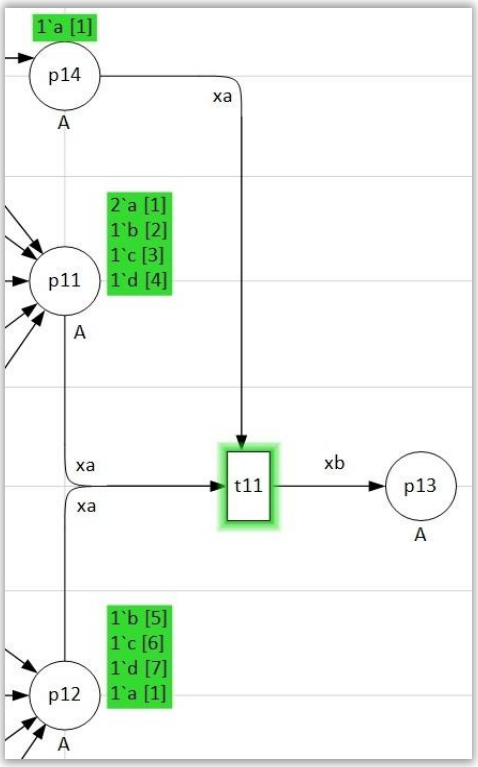
Po přesunu všech možných značek vypadá situace následovně: V místě *p11* jsou k dispozici stále ještě dvě značky a místo *p12* disponuje jednou značkou. Přechod i přes zaplněnost obou vstupních míst není proveditelný, protože již není možné odebrat stejnou značku z obou vstupních míst (*p11* a *p12*) pomocí jedné proměnné *xa*.



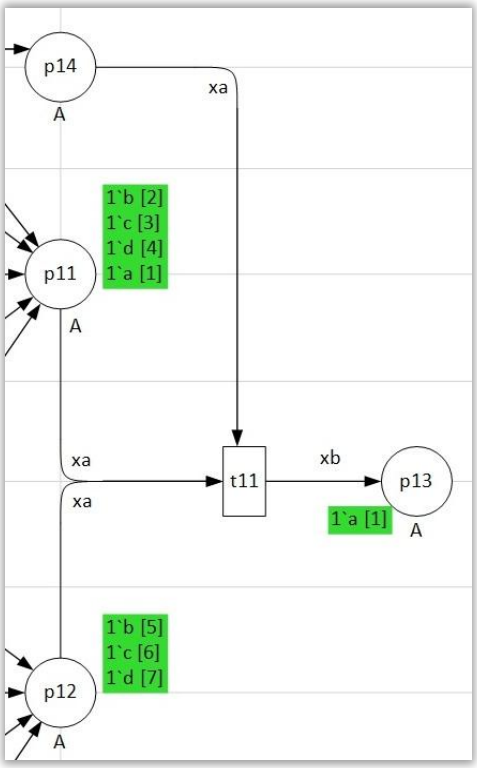
Obrázek 35 – Extrém př. 2 část 2

Extrémní síť 3

Tato síť je opět podobná předchozím dvěma, jen je na vstup přechodu t11 přivedena další hrana. Síť lze nalézt v přiloženém CD pod názvem *moje_sit3.cpn*. Následující dva obrázky ukazují znovu situaci před přenesení první značky a po ukončení evoluce.



Obrázek 36 – Extrém př. 3 část 1



Obrázek 37 – Extrém př. 3 část 2

Na druhém obrázku můžeme vidět opět ukončenou evoluci, protože místo p14 neobsahuje žádnou značku. Testovalo se zde přenášení značek pomocí jedné proměnné z více než dvou míst. Jak je tedy vidět na druhém obrázku, z míst p11, p12 a p14 se přenesla značka *a* do místa p13. Z pohledu instancí zde nebylo třeba vybírat, která instance se přenese, jelikož byly ve všech vstupních místech totožné.

6.3. Prověření generátoru náhodných čísel

Aby bylo možné uskutečnit náhodný výběr z několika možností, potřebujeme generátor náhodných čísel, pomocí kterého tuto volbu uskutečníme. Jazyk C#, kterým jsem programoval výslednou aplikaci, nabízí generátor, který lze vytvořit pomocí třídy *Random*. Tento generátor může pseudonáhodně generovat čísla v libovolném rozsahu hodnot díky metodě *Next*. Pro aplikaci je důležité, aby všechny možnosti, které budou generátorem voleny, měly stejnou šanci na výběr. Proto jsem sepsal jednoduchý program, který tento generátor otestuje.

Obsahem programu je generování pěti milionů čísel v rozmezí 0 – 9 a následný výpis, který informuje o počtech jednotlivých čísel a jejich procentuálním zastoupení v rámci všech vygenerovaných čísel. Program má následující zdrojový kód.

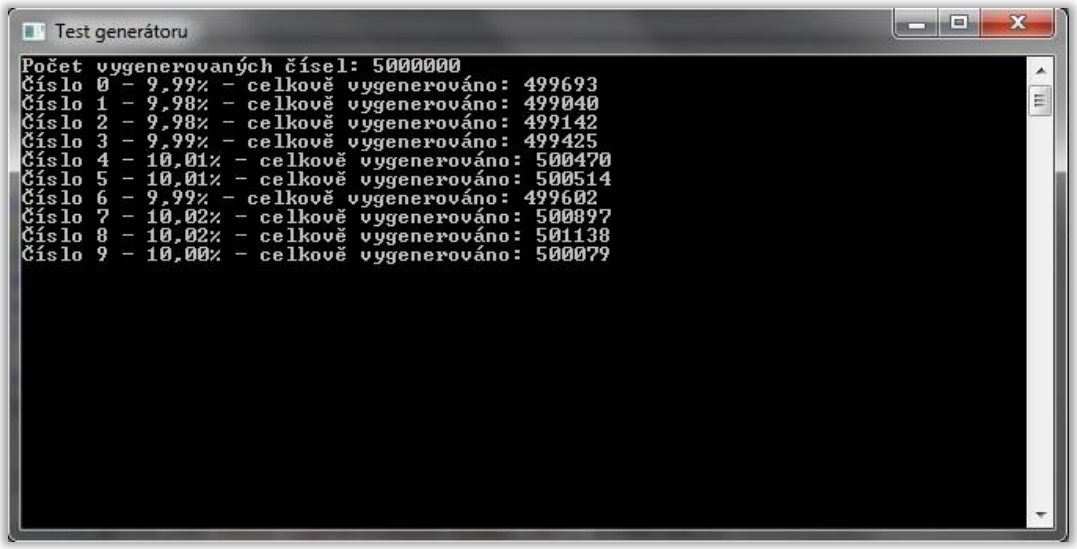
```
using System;

namespace TestGeneratoru
{
   class Program
   {
      static void Main()
      {
         Random generator = new Random();
         double[] pole = new double[10];
         const int pocetCisel = 5000000;
         Console.Title = "Test generátoru";

         for (int i = 0; i < pocetCisel; i++)
         {
            int cislo = generator.Next(0, 10);
            pole[cislo]++;
         }

         Console.WriteLine("Počet vygenerovaných čísel: {0}", pocetCisel);
         for (int i = 0; i < pole.Length; i++)
         {
            Console.WriteLine("Číslo {0} - {1} - celkově vygenerováno: {2}",
                i, (pole[i] / pocetCisel).ToString("P"), pole[i]);
         }
         Console.ReadKey();
      }
   }
}
```

Po spuštění se tedy vše zpracuje a jeden z možných výsledků můžeme vidět na následujícím obrázku. Maximální rozdíl mezi počty vygenerovaných čísel je podle výpisu 0,04%. Z tohoto by se dalo usuzovat, že použitý generátor splňuje potřebné nároky a dává všem možnostem stejnou pravděpodobnost výběru.



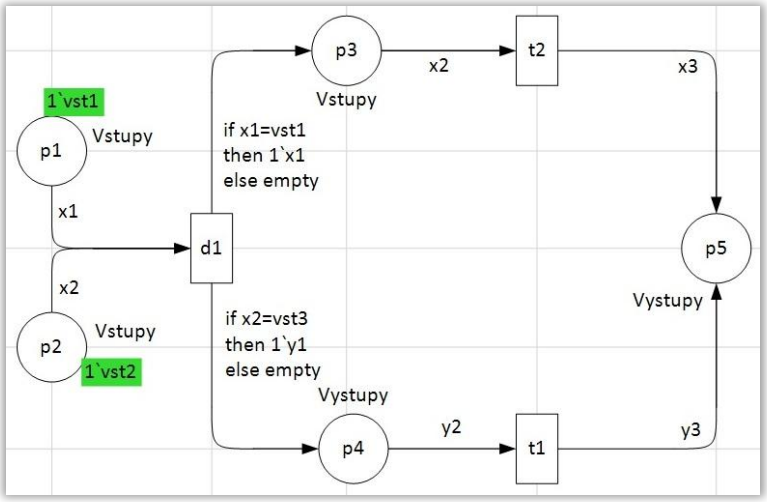
Obrázek 38 – Výpis při testování generátoru

6.4. Otestování verifikace

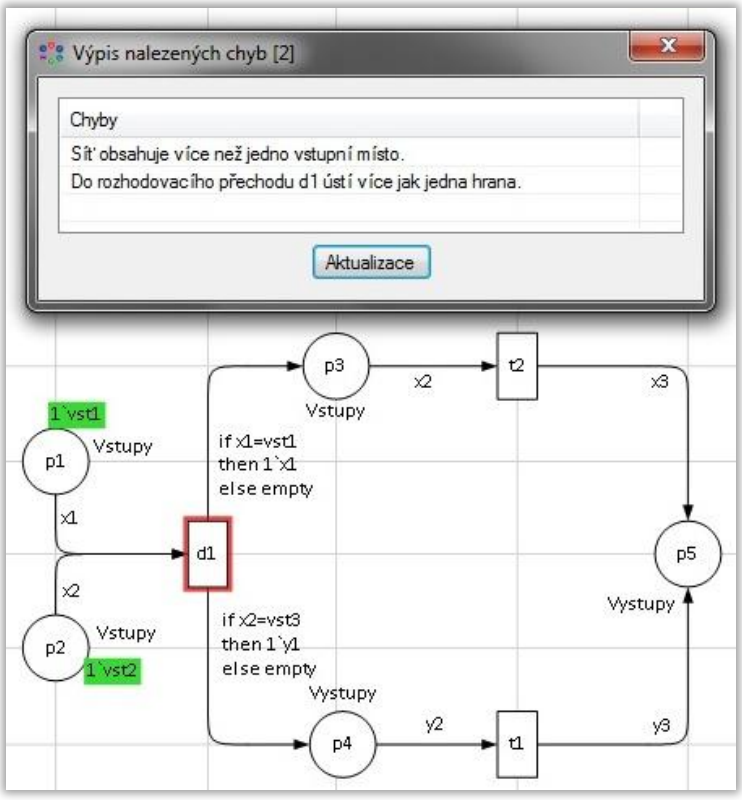
Testům podlehla i samotná verifikace. Tato podkapitola je zaměřena právě na tyto testy, které budou prezentovány několika ilustračními příklady s ukázkou výpisu chyb. Všechny zde testované sítě jsou součástí přiloženého CD.

Test verifikace příklad1

První testovací síť je na první pohled chybně postavená, protože obsahuje dvě vstupní místa. Verifikace však najde i druhou chybu, která z tohoto obrázku není hned zřejmá.



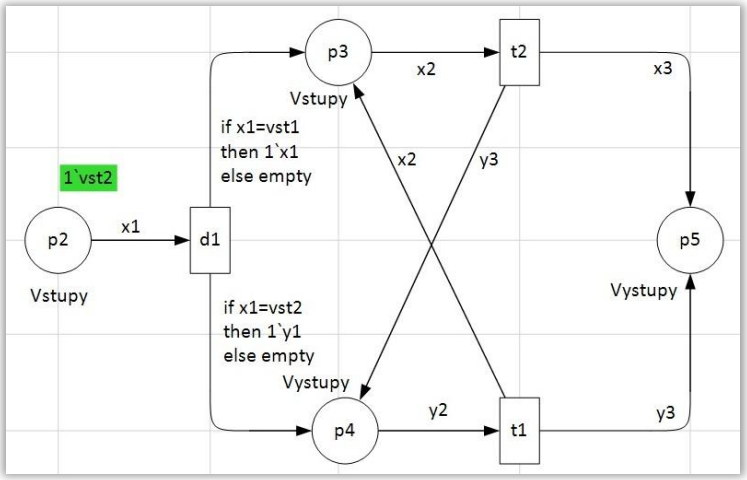
Obrázek 39 – Síť 1 před verifikací



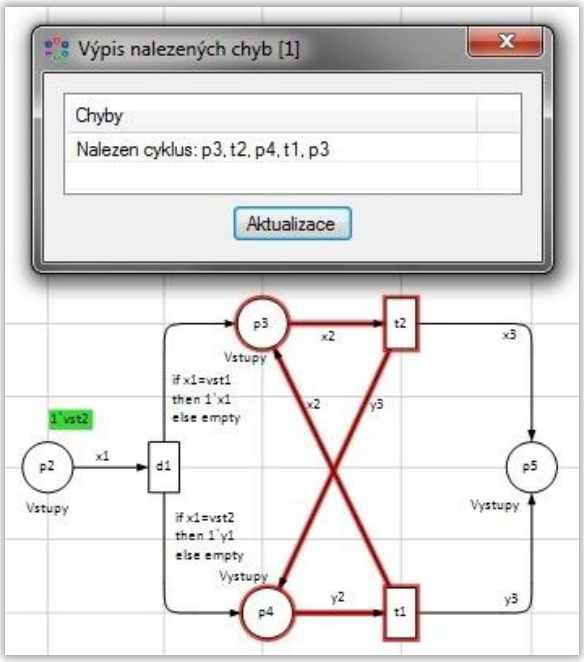
Obrázek 40 – Síť 1 po verifikaci

Test verifikace příklad 2

V druhé testovací síti už problém se vstupními místy není, ale vznikla tu situace vhodná pro testování algoritmu, který hledá cykly.



Obrázek 41 – Síť 2 před verifikací

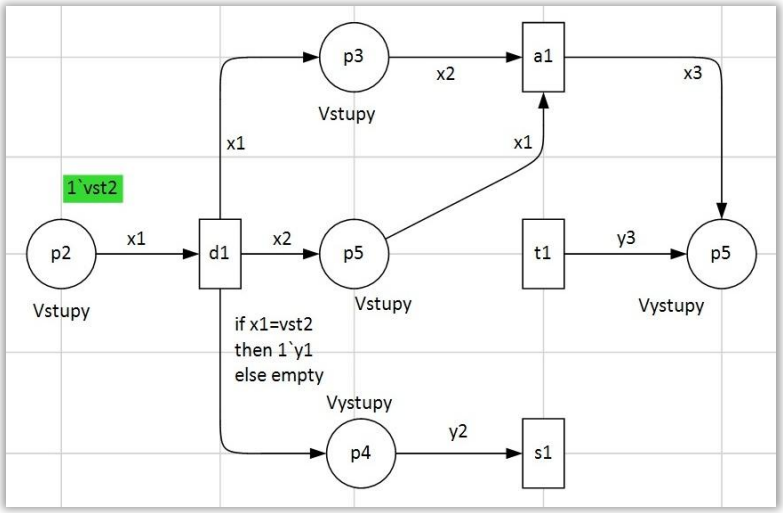


Obrázek 42 – Síť 2 po verifikaci

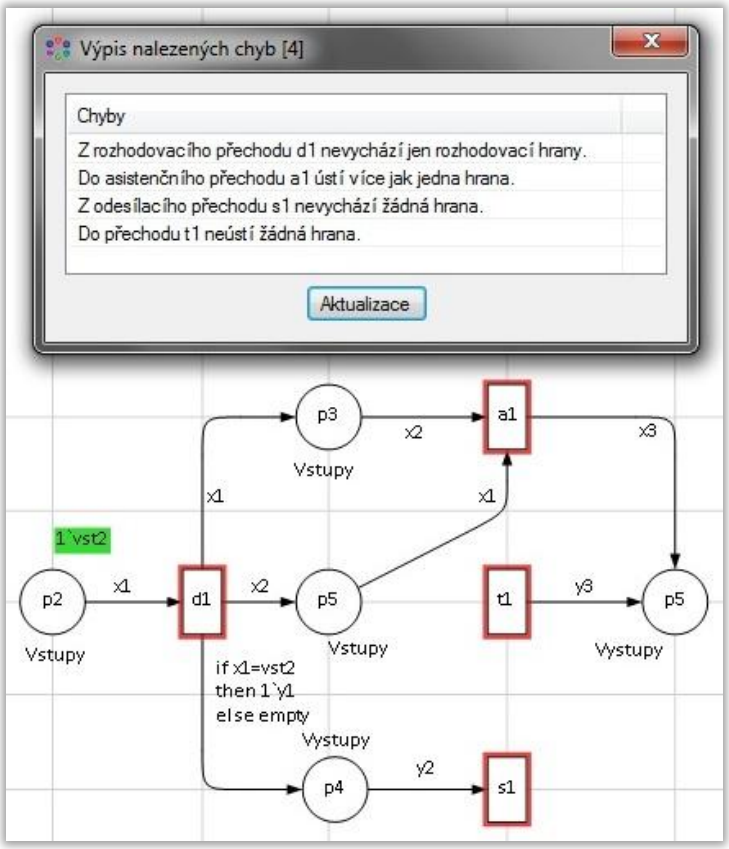
Z druhého obrázku je zřejmé, že algoritmus funguje a verifikace nejen že vypíše vrcholy obsažené v cyklu, ale také cyklus stejně jako většinu ostatních chyb pro lepší názornost zvýrazní červeným podbarvením.

Test verifikace příklad 3

Další test se bude věnovat verifikaci přechodů. Jelikož ABA-CPN rozlišuje mezi čtyřmi druhy, kde má každý přesně definované podmínky vrcholových okolí, tak tyto podmínky v následující síti otestujeme.



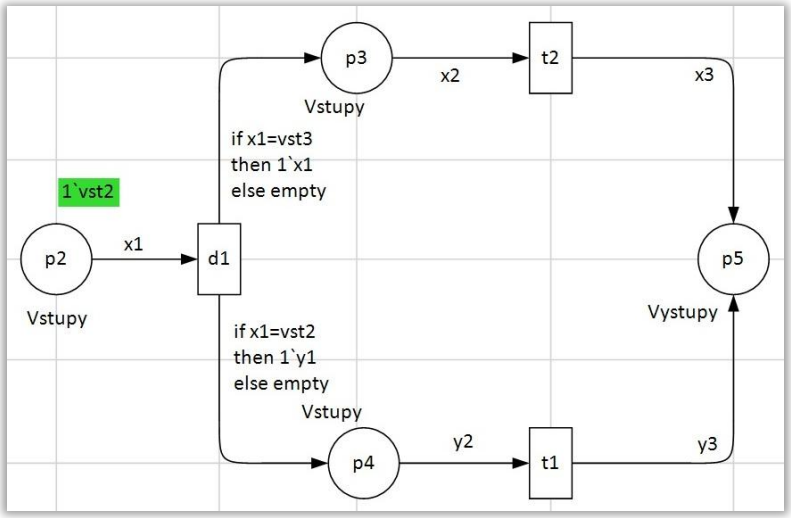
Obrázek 43 – Síť 3 před verifikací



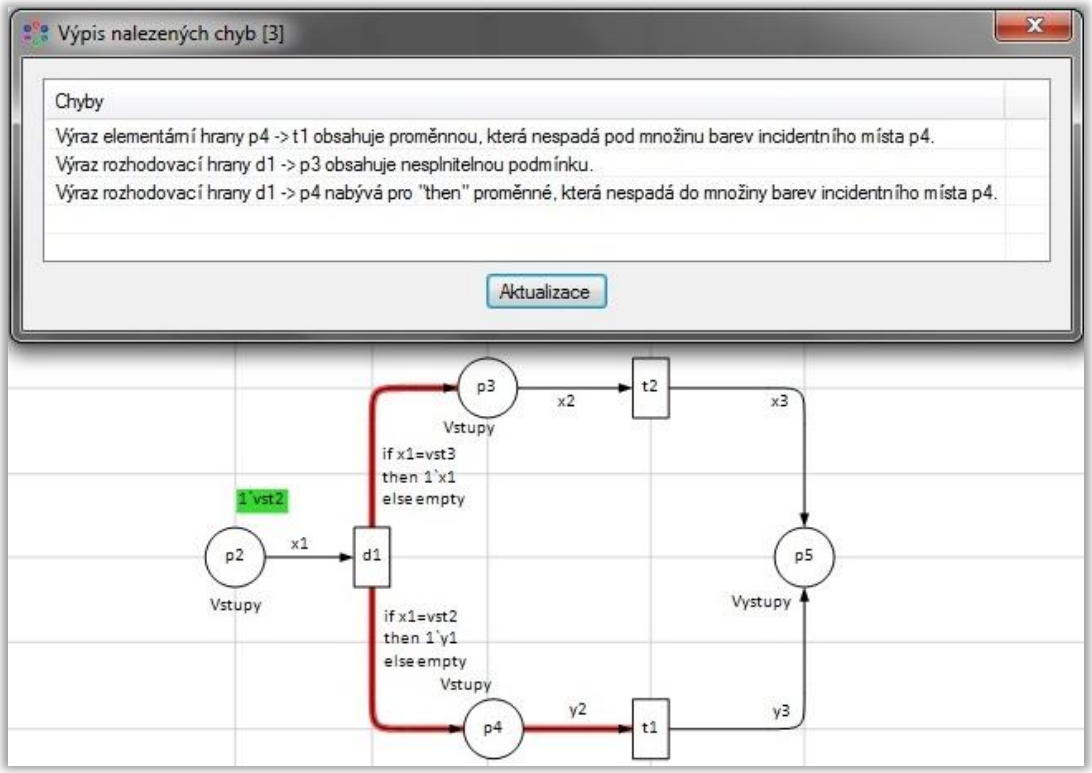
Obrázek 44 – Síť 3 po verifikaci

Test verifikace příklad 4

Poslední ukázka testu je zaměřena na hrany resp. na hranové výrazy. ABA-CPN Editor sice pro tvorbu těchto výrazů vytváří filtry tak, aby se těmto chybám předcházelo, ale i to se dá obejít.



Obrázek 45 – Síť 4 před verifikací



Obrázek 46 – Síť 4 po verifikaci

Pro pochopení nalezených chyb je potřeba říci, že proměnné *y1* a *y2* jsou definovány na množině barev *Vystupy* a množina barev *Vstupy* obsahuje pouze dva prvky *vst1* a *vst2*.

7. Závěr

Tato diplomová práce si především kladla za cíl vytvořit funkční aplikaci, kterou by bylo možné vytvářet ABA-CPN, tuto síť následně podrobit verifikaci a hlavně vytvořit interpreta této podtřídy, kterým by bylo možné v rámci stejné aplikace provádět evoluci sítě. Dalším požadavkem byla integrace nástroje pro výpočet stavového prostoru dané podtřídy ABA-CPN, jenž vytvářel v rámci své diplomové práce kolega Jiří Engliš. Celková aplikace měla pracovat s xml soubory, jež standardně využívá CPN Tools.

V úvodních kapitolách byl proveden přehled základních pojmů z oblasti modelování a simulace a následně byla popsána ABAsim architektura i s jejím nejdůležitějším prvkem, tj. agentem. Poté byla stručně uvedena problematika Petriho sítí a následně byla definována barvená Petriho síť a popsáno její chování. Nejdůležitější teoretickou částí byl bezpochyby popis ABA-CPN, který spojoval předešlá témata, neboť ABA-CPN je podtřídou barvené Petriho sítě a jejím účelem je realizace interní logiky některých komponent agenta v rámci ABAsim architektury. Součástí definice ABA-CPN byly i konvence pro popis, které jsou důležité především pro interpreta, jenž díky těmto popisům může provádět evoluci korektně.

V kapitolách, jež byly zaměřeny na praktickou část, byly nejprve představeny editory pro tvorbu ABA-CPN resp. CPN Tools a ABA-CPN Editor, který je součástí výsledné aplikace. Byly zde popsány možnosti výměny souborů mezi těmito editory a následně více přiblížen ABA-CPN Editor a jeho nejdůležitější součásti. Proběhlo zde i představení důležitých datových struktur, které jsou použity aplikací, a popis jejich používání. Následně byly ukázány i hledané chyby v rámci verifikace a použitý algoritmus pro hledání cyklů. Poslední kapitola, která se týkala praktické části, byla zaměřena na provádění testů jednotlivých částí aplikace. Byl zde testován interpret, a to i pro velmi nestandardní sítě, generátor náhodných čísel a nakonec proběhlo několik ukázek testů verifikace.

Přínos této práce spočívá ve vytvoření komplexní aplikace, která je schopná vytvořit ABA-CPN, tuto síť verifikovat, provádět na ní testy prostřednictvím evoluce sítě a také spočítat a ukázat stavový prostor sítě. Všechny tyto funkcionality jsou tedy dostupné v rámci jedné aplikace, která navíc pracuje se standardním formátem souborů pro CPN. Zbývá jen dodat, že veškeré předem stanovené cíle byly splněny.

8. Seznam bibliografických citací

[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 simulacia 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 CPN Tools.* K. Jensen (Ed.). Aarhus: University of Aarhus (Denmark), 2008. pp. 47-62. ISSN 0105-8571.

[4] ADAMKO, N., KAVIČKA, A., KLIMA, V.: Agent based simulation of transportation logistic systems. *DAAAM International Scientific Book 2007*, Chapter 36, B. Katalinic (Ed.). Vienna: DAAAM International, 2007. pp. 407-422. ISBN 3-901509-60-7.

[5] VESELÝ, P.: *Interpret barvené Petriho sítě v rámci ABAsim architektury simulačních modelů*. [s.l.], 2007. Diplomová práce. Univerzita Pardubice.

[6] ŽARNAY, M.*: Systém na podporu rozhodovania pre riadenie dopravných procesov*. Žilina, 2007, 121 s, Dizertační práce na Fakultě řízení a informatiky Žilinské univerzitě v Žilině, Vedoucí dizertační práce prof. Ing. Petr Cenek, CSc.

[7] *CPN Tools home page* [online]. [cit. 2011-07-01]. Dostupné z WWW: <http://cpntools.org/>.

[8] *Wikipedia : Otevřená enciklopedie* [online]. [cit. 2011-07-01]. Prohledávání do hloubky. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Prohledávání_do_hloubky>.

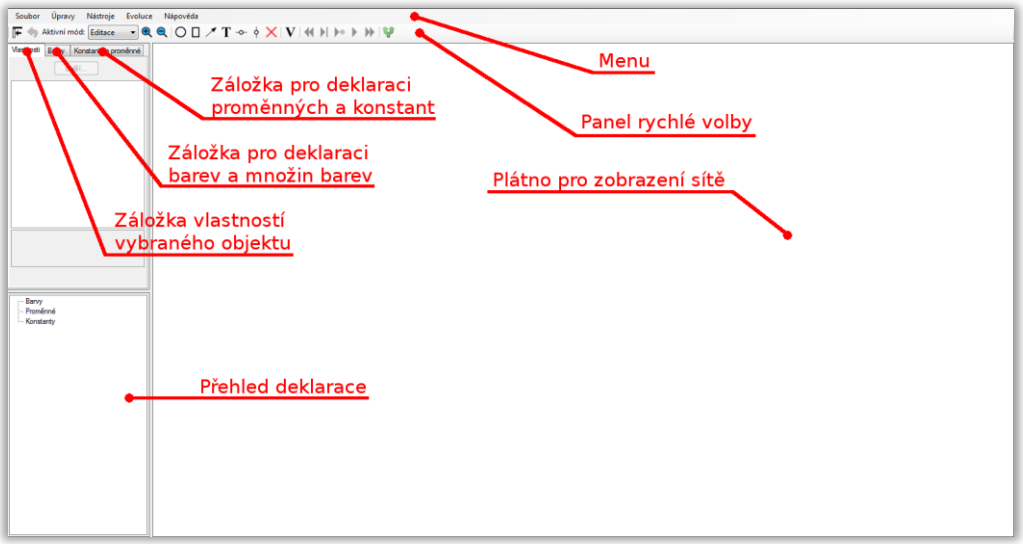
[9] PETZOLD, CH.: *Programování Microsoft Windows v jazyce C#*. Praha: SoftPress, 2003. 1188 s. ISBN: 80-86497-54-2.

9. Přílohy

Příloha A. Manuál k obsluze ABA-CPN Editoru

Uživatelské rozhraní

Rozložení jednotlivých panelů je vidět na následujícím obrázku. Dále budou detailně popsány jednotlivé panely a jejich možnosti.



Obrázek 47 – Uživatelské rozhraní ABA-CPN Editoru

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*, které se po potvrzení výběru souboru pokusí otevřít a zobrazit síť. Klávesová zkratka *ctrl+O*.
- **Uložit** – Pokud již byla otevřená síť 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). Pamatuje si jen posledních deset změn. 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 s kurzorem myši a je tak možné vybrat pozici vkládaného místa. Po nalezení vhodného místa stiskněte pro vložení 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 se vytvoří hrana tak, že se nad výchozím vrcholem zamýšlené hrany stiskne a drží levé tlačítko myši. Potom se za stálého držení tohoto tlačítka přesune kurzor myši nad cílový vrchol, kde se držené tlačítko pustí. 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*.

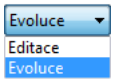
- **Automatické provedení přechodu** – Po aktivaci je možno kliknutím levého tlačítka myši na proveditelný přechod provést náhodně právě tento přechod. Kliknutím levého tlačítka myši kamkoli jinam způsobí náhodné vybrání proveditelného přechodu a jeho provedení. Klávesová zkratka *ctrl+A*.
- **Kontrolované provedení přechodu** – Po aktivaci je možno kliknutím levého tlačítka myši na proveditelný přechod provést právě tento přechod. Následně se otevře dialogové okno, s možností nastavit jednotlivé proměnné a konstanty účastnící se provádění přechodu. Přesnější popis dialogového okna bude uveden dále. 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 dispozici 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:

- – Schová nebo zobrazí celý panel vlevo od *Plátna*.
- – Stejná akce jako *Úpravy → Zpět*.
- 

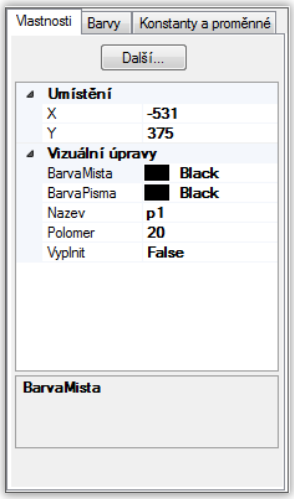
– Přepíná mezi režimy editoru. Do režimu *Evoluce* se lze úspěšně přepnout jen tehdy, když síť projde bez nalezených chyb verifikací. Režimy editoru budou vysvětleny níže.
- – Stejná akce jako *Úpravy → Přiblížit*.
- – Stejná akce jako *Úpravy → Oddálit*.
- – Stejná akce jako *Nástroje → Vlož místo*.
- – Stejná akce jako *Nástroje → Vlož přechod*.
- – Stejná akce jako *Nástroje → Vlož hranu*.
- – Stejná akce jako *Nástroje → Vlož textovou poznámku*.
- – Stejná akce jako *Nástroje → Vlož horizontální vodítko*.

-  – Stejná akce jako *Nástroje → Vlož vertikální vodítko*.
-  – Stejná akce jako *Nástroje → Odeber element*.
-  – Stejná akce jako *Nástroje → Verifikace*.
-  – Stejná akce jako *Evoluce → Zpět na začátek*.
-  – Stejná akce jako *Evoluce → Automatické provedení přechodu*.
-  – Stejná akce jako *Evoluce → Kontrolované provedení přechodu*.
-  – Stejná akce jako *Evoluce → Spustit evoluci*.
-  – Stejná akce jako *Evoluce → Rychlé provedení přechodu*.
- – Otevře *ABA-CPN stavové prostory* s aktuální sítí.

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 libovolné místo. S hranami se takto ale hýbat nedá, a tak místo posunu celé hrany se na místě chycení hrana ohýba. Pokud je hrana chycena za již vytvořený ohyb, hýbeme právě s tímto ohybem a nový se nevytváří. Jak již bylo zmíněno, 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 zakrývaly 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, můžeme na ní vidět a většinou i měnit všechny vlastnosti vybraného elementu, ale o této záložce později. Toto označování funguje pouze v režimu *Editace*.

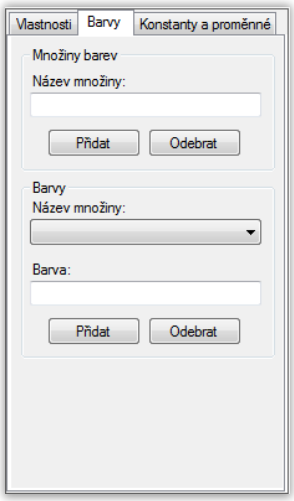
Záložka vlastností vybraného objektu



Obrázek 48 – Záložka vlastností

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*. Tato dialogová okna budou probrána 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, jenž potom tento soubor není schopen otevřít, tak je tu tento filtr zaveden.

Záložka pro deklaraci barev a množin barev



Obrázek 49 – Záložka pro deklaraci barev a množin barev

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

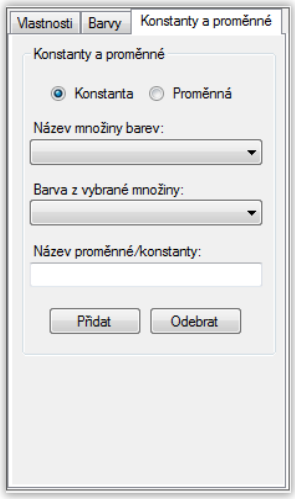
První sekce s názvem *Množiny barev* slouží pro manipulaci s celými množinami barev. Pro vložení nové množiny stačí do textového pole s označením *Název množiny* zapsat název vkládané množiny barev a následně stisknout tlačítko *Přidat*. Pro odebrání množiny barev se musí její název objevit ve stejném 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.

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 proto se

odeberou. Podobně jsou na tom i konstanty, o jejichž odebrání vám zmíněné informativní okno také podá informaci.

Záložka pro deklaraci proměnných a konstant



Obrázek 50 – 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.

Přehled deklarace

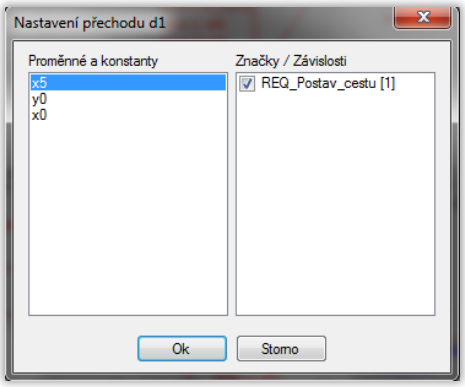
V tomto panelu je znázorněna veškerá deklarace v podobě stromových struktur. První kořen stromu *Barvy* obsahuje na další úrovni názvy jednotlivých deklarovaných množin barev. Další úroveň pod množinami obsahuje už konkrétní barvy z dané množiny barev. Druhý kořen stromu *Proměnné* obsahuje na další 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, vkládání ani odebírání elementů sítě, nelze označovat elementy a ani nelze s většinou elementů hýbat.

Dialogová okna

1. Dialogové okno pro nastavení provedení přechodu.



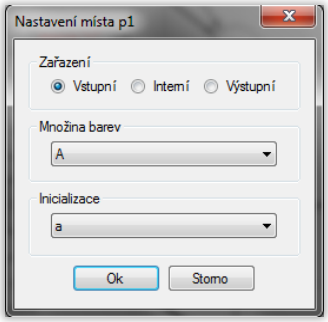
Obrázek 51 – Okno nastavení provedení přechodu

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 dialog zavře bez provádění 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. Na místě instance může také být název proměnné nebo konstanty. To značí, že instance bude od této konstanty či proměnné přebrána. Poslední možností, jež se může na místě instance objevit, je slovo *nová*, což značí, že se zde vytvoří pro danou barvu nová instance.

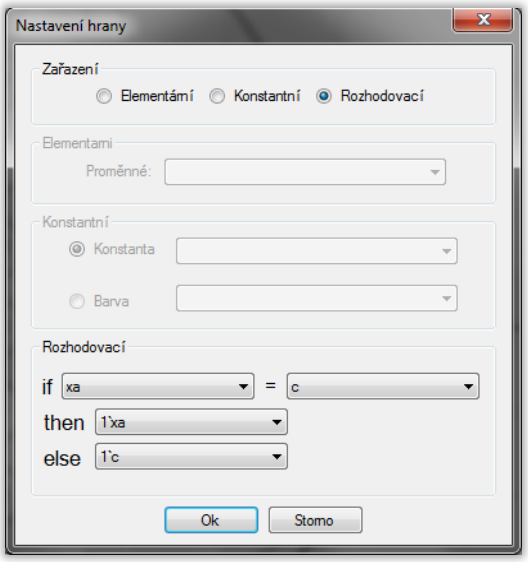
2. Dialogové okno pro nastavení místa



Obrázek 52 – Okno 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.

3. Dialogové okno pro nastavení hrany



Obrázek 53 – Okno 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.

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.

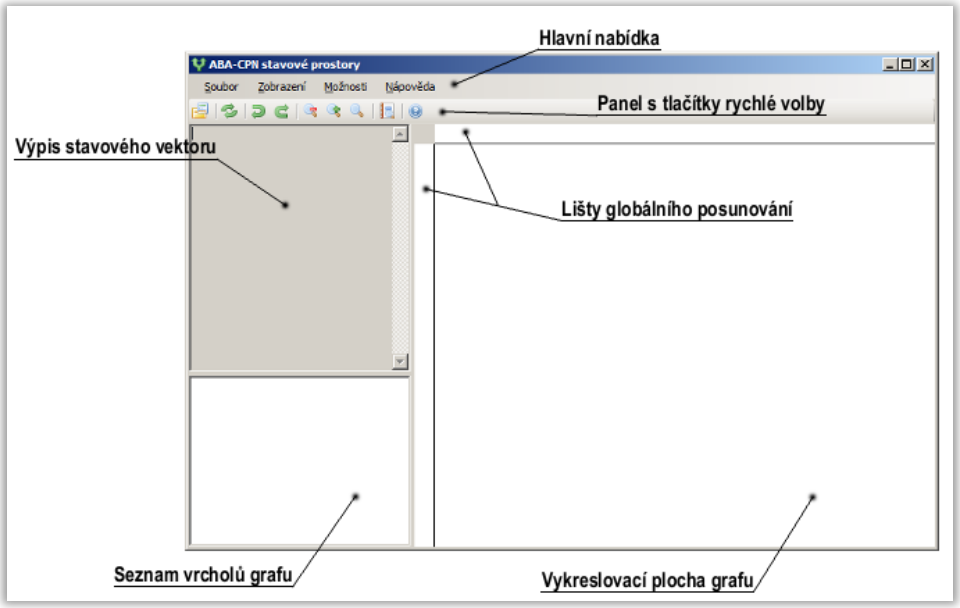
Poslední sekce nastavuje rozhodovací hranu. Je postavena pomocí ComboBoxů, které jsou uspořádány tak, že je vidět, kterou část rozhodovacího výrazu vyplňují. ComboBox umístěný za *if* obsahuje jen proměnné, umístěné na vstupu incidentního rozhodovacího přechodu. ComboBox umístěný za znakem = obsahuje 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*.

Příloha B. Manuál k obsluze ABA-CPN stavové prostory

Uživatelské rozhraní

Rozhraní aplikace vypadá následovně:



Obrázek 54 – Uživatelské rozhraní ABA-CPN stavové prostory

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*).

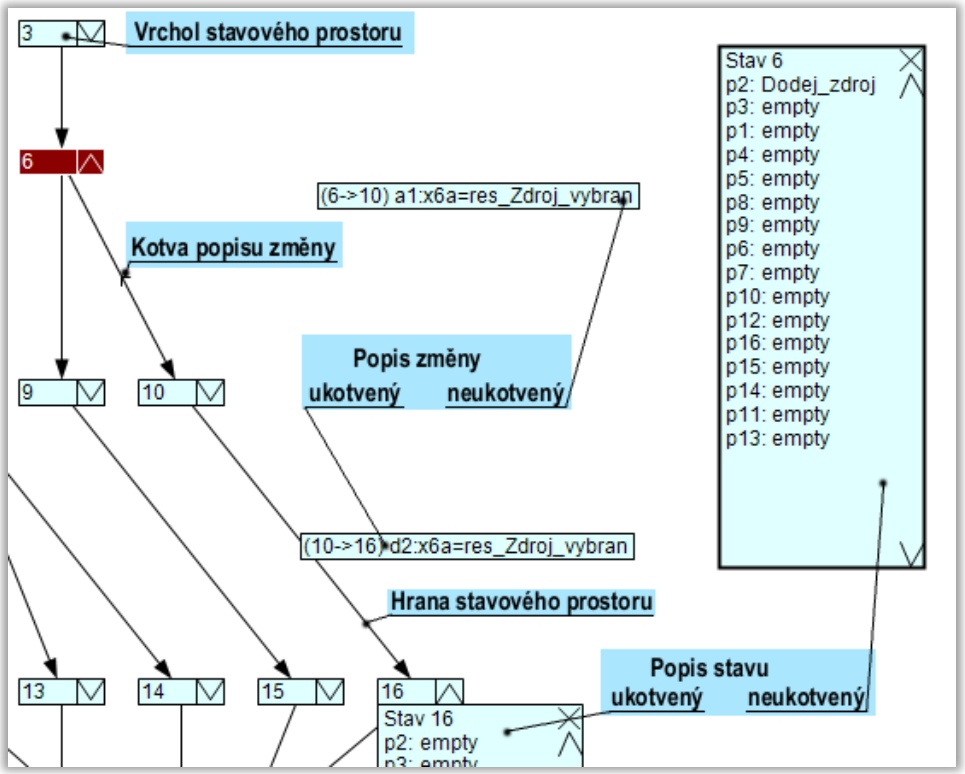
Na vykreslovací plochu grafu se vykresluje graf stavového prostoru.

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 vypadají následovně:



Obrázek 55 – Prvky grafu stavového prostoru

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 "Λ" posouvá řádky výpisu o řádek výše a tlačítko "V" 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, odpálený 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.

Barvy a velikosti prvků lze měnit, viz *Nastavení*.

Hlavní nabídka

Nabídka *Soubor* obsahuje hlavně operace pro práci s okolím aplikace. Nabízené operace jsou následující:

- **Otevřít uložený prostor** – Zobrazí otevírací dialog, ve kterém vyberete soubor s příponou *stsp*. Z vybraného souboru se pak aplikace pokusí rekonstruovat předem vypočítaný graf stavového prostoru. Zároveň provede obnovení zobrazení. Klávesová zkratka *ctrl+alt+O*.
- **Uložit prostor** – Zobrazí ukládací dialog, ve kterém vyberete soubor s příponou stsp. Do vybraného souboru se pak aplikace pokusí zapsat aktuálně vypočítaný graf stavového prostoru. Klávesová zkratka *ctrl+S*.
- **Přepočítat síť** – Přinutí aplikaci, aby přepočítala stavový prostor sítě. Zároveň provede obnovení zobrazení. Je-li síť načtena ze souboru, před výpočtem se načte znovu. Klávesová zkratka *ctrl+R*.
- **Vytisknout** – Zobrazí dialog nastavení tiskárny. Na základě nastavení se pak pokusí vytisknout aktuálně vypočítaný graf stavového prostoru. Klávesová zkratka *ctrl+P*.
- **Konec** – Ukončí aplikaci. Klávesová zkratka *ctrl+Q*.

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. Klávesová zkratka *ctrl+Z*.
- **Opakovat** – Opakuje poslední změnu grafu stavového prostoru – polohu prvků nebo jejich viditelnost. Klávesová zkratka *ctrl+Y*.
- **Přiblížit** – Přiblíží graf (zvětšení). Klávesová zkratka *ctrl+OemPlus*.
- **Oddálit** – Oddálí graf (zmenšení). Klávesová zkratka *ctrl+OemMinus*.
- **Obnovit** – Obnoví přiblížení na původní (1:1) a nastaví souřadnice levého horního rohu na [0,0]. Klávesová zkratka *ctrl+O*.
- **Export do PNG** – Otevře dialog pro ukládání, ve kterém vyberete soubor s příponou png. Do vybraného souboru se pak aplikace pokusí zapsat obrázek typu PNG, který zobrazuje aktuálně vypočítaný graf stavového prostoru. Klávesová zkratka *ctrl+E*.

Nabídka *Možnosti* 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. Klávesová zkratka *ctrl+alt+S*.
- **Předvolby** – Zobrazí dialog, který obsahuje možnosti pro nastavení velikostí a barev prvků grafu stavového prostoru.

Akční tlačítka

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

-  – stejná akce jako *Soubor → Přepočítat síť*,
-  – stejná akce jako *Zobrazení → Zpět*,
-  – stejná akce jako *Zobrazení → Opakovat*,
-  – stejná akce jako *Zobrazení → Oddálit*,
-  – stejná akce jako *Zobrazení → Přiblížit*,
-  – stejná akce jako *Zobrazení → Obnovit*,
-  – stejná akce jako *Možnosti → Zobrazit report*,
-  – stejná akce jako *Nápověda → Obsah*.

Popis těchto operací lze nalézt v hlavní nabídce.

Změny pohledu na graf

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

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 grafu stavového prostoru

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.

Aplikace umožňuje několik způsobů, jak vybrat prvek. 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.

Existují dvě možnosti, jakými aplikace umožňuje posunování prvků. Při posunu se uplatňuje neviditelná mřížka, ke které se prvky přichytávají.

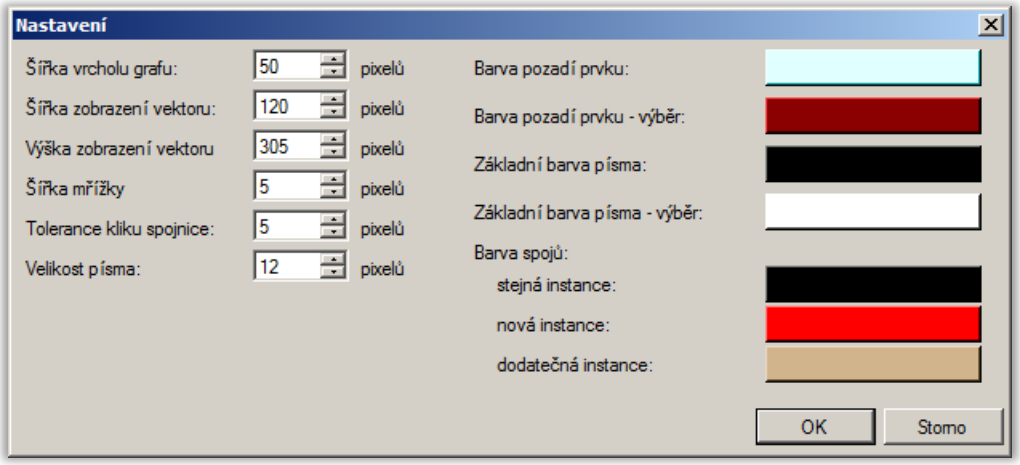
- Posunování vybraných prvků. Pokud bez stisknutých modifikačních kláves stisknete levé tlačítko myši nad libovolným vybraným prvkem a započnete táhnutí, zobrazí se obrysy všech vybraných prvků. Tyto obrysy vyznačují novou pozici vybraných prvků. Po uvolnění tlačítka myši se prvky na označená místa přesunou. 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 hranicí, 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.

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 po tom, co je stavový prostor dopočítán.

Nastavení aplikace

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



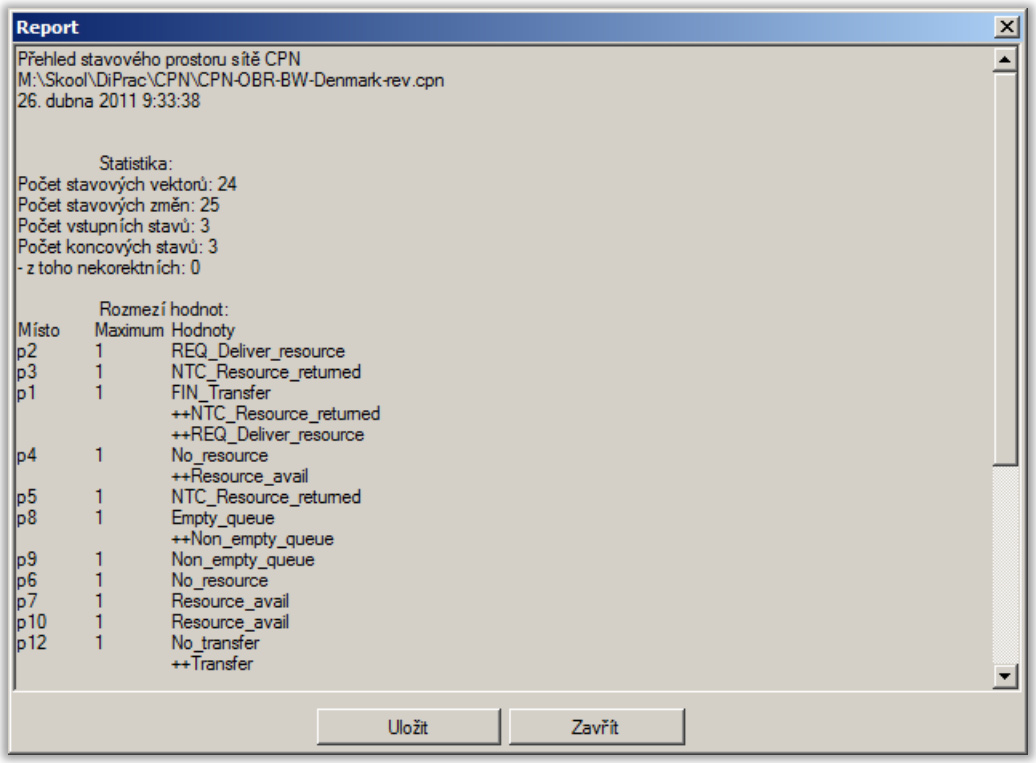
Obrázek 56 – Nastavení ABA-CPN stavové prostory

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 57 – Zpráva o stavovém reportu

Dialog se skládá z needitovatelného textového pole, které obsahuje zprávu, tlačítko, které umožňuje uložení zprávy, a tlačítko, 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 odpáleny 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.