Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Editor Petriho sítí

2019

Roman Wehmhőner

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Roman Wehmhőner
Název práce:	Editor Petriho sítí
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	36
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Roman Wehmhőner
Title:	Petri nets editor
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	36
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem bakalářské práce bylo vytvořit editor Petriho sítí umožňující jednoduché a pohodlné ovládání. Editor také obsahuje základní nástroje pro analýzu Petriho sítí.

Synopsis

The goal of bachelor thesis is to make editor for Petri nets with simple convenient control. Editor contains basic analysis tools too.

Klíčová slova: Petriho síť; Editor

Keywords: Petri net; Editor

Tímto bych chtěl poděkovat vedoucímu bakalářské práce panu Mgr. Petrovi Osičkovi, Ph.D. za hodnotné rady při vývoji editoru.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Petriho sítě

Tato kapitola byla inspirovaná a čerpala informace z knihy Understanding petri nets[1].

1.1 Základní popis

Petriho síťe jsou matematickým nástrojem pro modelování a simulaci paralelních procesů a jejich sychronizaci. Jsou tvořené místy, přechody a hranami. Každá hrana vždy propojuje jeden přechod s jedním místem.

Definice 1 (Petriho síť)

$$N = \langle P, T, A, M_0 \rangle$$

- N je Petriho sítí.
- P je konečná množina míst.
- T je konečná množina přechodů.
- A je konečná množina hran. $A \subseteq ((P \times T) \cup (T \times P)) \times \mathbb{N}_0$
  kde číslo symbolizuje násobek kolik značek hrana „přesune“.
- $M_0 : P \to \mathbb{N}_0$ je počáteční ohodnocení míst sítě (zkráceně ohodnocení), kde pro každé místo $p \in P$ existuje počet jeho značek $M_0(p) = \mathbb{N}_0$.

Pro odkazovaní na jednotlivé členy prvků z množiny hran $a \in A$ budeme používat notaci $P(a)$ pro odkázání na místo hrany a, $T(a)$ pro odkázání na přechod a $AM(a)$ pro odkaz na násobek.

Každý přechod t může mít „přiřazený“ libovolný počet hran $a \in A_t$, kde každá hrana a je propojením přechodu t s některým z míst $p \in P$.
Hrany přechodu t můžeme rozlišit na hrany směrující do přechodu

$${}^{\rightarrow}t = \{a \in A | a \in (P \times T \times \mathbb{N}_0) \wedge t = T(a)\}$$

a hrany směřující z přechodu (do místa)

$$t^{\rightarrow} = \{a \in A | a \in (T \times P \times \mathbb{N}_0) \wedge t = T(a)\}$$

dohromady pak všechny hrany přechodu t jsou spojením těchto dvou množin

$$ArcesOfTransition(t, A) = A_t = ({}^{\rightarrow}t \cup t^{\rightarrow}) \quad \subset A$$

Mezi libovolným přechodem $t \in T$ a libovolným místem $p \in P$ může existovat **maximálně jedna** hrana $a_{pt} \in {}^{\rightarrow}t$ a **maximálně jedna** hrana $a_{tp} \in t^{\rightarrow}$ (dohmady tedy maximálně 2 hrany, jedna **z** a druhá **do** přechodu).

Aktuální stav Petriho sítě neboli ohodnocení M je funkce přiřazující každému místu $p \in P$ Petriho sítě počet značek

$$(\forall p \in P) M(p) \in \mathbb{N}_0$$

Počáteční ohodnocení Petriho sítě se značí M_0.

Pro libovolné ohodnocení M je přechod $t \in T$ označený jako **povolený**, pokud všechny hrany směřující do přechodu ${}^{\rightarrow}t$ splňují svou podmínku tzn. hrana splňuje svoji podmínku, pokud místo ze kterého vychází má vyšší nebo stejné ohodnocení (v daném M) než je násobek hrany AM

$$IsEnabled(P, t, A, M) = (\forall a \in {}^{\rightarrow}t) M(P(a)) \geq AM(a)$$

Pak si můžeme ještě definovat množinu všech povolených přechodů pro zadané ohodnocení

$$EnabledTransitions(P, T, A, M) = \{t \in T | IsEnabled(P, t, A, M)\}$$

Pokud je přechod t v ohodnocení M Petriho sítě **povolený**, znamená to že může dojít k **aktivování** tohoto přechodu, čímž dojde ke změně aktuálního ohodnocení z M do ohodnocení M' tak, že pro každé místo $p \in P$ a hrany $A_t \subset ArcesOfTransition(t, A)$ spojující p s t že nové ohodnocení v místě $M'(p)$ je sumou násobků hran $\sum_{a \in A_t} AM(a)$ a původního hodnocení $M(p)$

$$FireTransition(P, t, A, M) = function\ M'$$

Výsledné ohodnocení M' je pak pro každé místo p definováno

$$M'(p) = M(p) + \sum_{a \in \{a_{tp} \in ArcesOfTransition(t,A) \,|\, P(a_{tp}) = p\}} AM(a)$$

Tuto změnu značíme $M \rightarrow^t M'$.

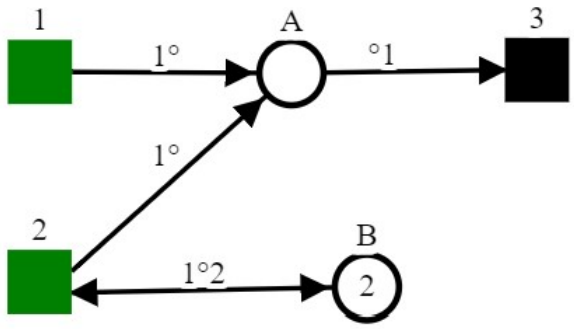
Ohodnocení M' je označené jako **dosažitelné** z ohodnocení M, pokud existuje sekvence přechodů taková, že jejich postupným **aktivováním** z ohodnocení M vznikne ohodnocení M'. Ohodnocení M' je dostupné z ohodnocení M pak značíme $M \rightarrow^* M'$.

1.2 Vizuální zobrazení sítě

Pro Petriho síť existuje nejenom matematické zobrazení, ale i v praxi více využívané grafické zobrazení. V grafickém zobrazení kolečka symbolizují místa Petriho sítě a číslo v kolečku udává počet značek. Přechody jsou symbolizované čtverci. V editoru je čtverec zelený, pokud je přechod povolený. Pokud má v sobě čtverec symbol ϵ, znamená to že je přechod určený pro komunikaci sítě s okolním prostředím. Na editor nemá žádný vliv jestli je přechod označený jako ϵ a tak tedy toto označení je jen pro jednodušší orientaci v síti. Kolečka i čtverce mají pak nad sebou značení konkrétního místa/přechodu které symbolizují. Nakonec

samotné hrany jsou symbolizované šipkami které jsou popsané násobkem kolik hrana „přesune“ značek. Popis hran je symbolizován dvěmi čísly oddělenými kolečkem. Číslo před kolečkem značí hodnotu hrany směrující z přechodu do místa, číslo za kolečkem značí hodnotu hrany směrující z místa do přechodu.

Na obrázku 1 můžeme vidět grafické vyobrazení jednoduché sítě s dvěmi místy, třemi přechody a čtyřmi hranami.



Obrázek 1: Příklad zobrazení jednoduché sítě

Již na první pohled si můžeme všimnou, že pro jednoduší rozlišení míst a přechodů jsou přechody značené čísli a místa písmeny. Tuto síť vyobrazenou na obrázku 1 bychom mohli matematickým zápisem zapsat jako síť $N = \langle P, T, A, M_0 \rangle$ kde
$P = \{a, b\}$
$T = \{1, 2, 3\}$
$A = \{\langle 1, a, 1\rangle, \langle 2, a, 1\rangle, \langle 2, b, 1\rangle, \langle b, 2, 2\rangle, \langle a, 3, 1\rangle\}$
$M_0(a) = 0; M_0(b) = 2$

V matematickém zápise sítě budeme místa značit malými písmeny (oproti editoru) abychom předešli případným nedorozuměním.

1.3 Zkrácený zápis ohodnocení sítě

Abychom se vyhnuli zdlouhavému psaní každého případu ohodnocovací funkce (např. $M(a) = 0; M(c) = 2; \cdots$), zavedeme si kratší zápis. Nejdříve seřadíme všechny místa podle jejich značení abecedně jakoby šlo o číselnou soustavu (s písmeny, bez čísel) neboli $a, b, c, \cdots, z, aa, ab, ac, \cdots$. Pak si z těchto míst uděláme uspořádanou n-tici jejich ohodnocení $\langle M(a), M(b), M(c), \cdots \rangle$. Tuto n-tici pak budeme používat jako kratší zápis ohodnocovací funkce:

$$M = \langle M(a), M(b), M(c), \cdots \rangle$$

Například pro $M'(a) = 3; M'(b) = 0; M'(c) = 5$ je krátký zápis $M' = \langle 3, 0, 5 \rangle$

1.4 Využití Petriho sítí

Petriho sítě se používají k analýze a modelování paralelních a distribuovaných systému, databázových systémů atd. a to ať už pro analýzu při vývoji softwaru a

nebo pro popis vnitřní struktury již hotového proprietárního softwaru umožnující lepší porozumění uživateli.

1.5 Graf dosažitelnosti

Graf dosažitelnosti je jeden z nejzákladnějších nástrojů pro analýzu Petriho sítí. Obsahuje vždy počáteční ohodnocení a všechny ohodnocení které jsou **dosažitelné** z počátečního ohodnocení, takovéto ohodnocení budeme zkráceně nazývat **dosažitelné ohodnocení**. Vrcholy grafu jsou jednotlivá ohodnocení a hrany grafu jsou značené přechody které jsou aktivované aby z počátečního ohodnocení vzniklo cílové.

Definice 2 (Graf dosažitelnosti)

$$RG = \{M, \langle M, T', M' \rangle\}$$

- RG je Graf dosažitelnosti
- M je Vrchol grafu který je zároveň konkrétní ohodnocení Petriho sítě
- $\langle M, T', M' \rangle$ je Hrana grafu která je změnou z hodnocení M libovolným přechodem $t \in T'$ ze kterého vzniká M'

1.5.1 Vlastnosti odvoditelné z Grafu dosažitelnosti

Z grafu dosažitelnosti Petriho sítě jsou odvoditelné tyto vlastnosti:

Definice 3 (Ohraničenost sítě)
Petriho síť je **ohraničená**, pokud je její graf dosažitelnosti konečný. Pokud existuje takové přirozené číslo n pro které v žádném dosažitelném ohodnocení nepřesahuje žádné místo svým ohodnocením číslo n a zvolíme n aby splňovalo tuto podmínku a zároveň bylo nejmenší možné, pak můžeme nazvat síť že je **ohraničená** číslem n.

Definice 4 (Konečnost sítě)
Petriho síť **skončí** za předpokladu že graf je konečný a zároveň neobsahuje žádné cykly. Neboli Petriho síť vždy po nějakém počtu kroků dojde do stavu, kdy žádný přechod není povolený.

Definice 5 (Vratnost sítě)
Petriho síť je **vratná**, pokud je její graf silně souvislý. Z každého dosažitelného ohodnocení je dosažitelné počáteční ohodnocení.

Definice 6 (Síť bez mrtvého bodu)
Petriho síť je **bez mrtvého bodu**, pokud z každého vrcholu grafu dosažitelnosti vede minimálně jedna hrana. Petriho síť má v každém ohodnocení povolený minimálně jeden přechod.

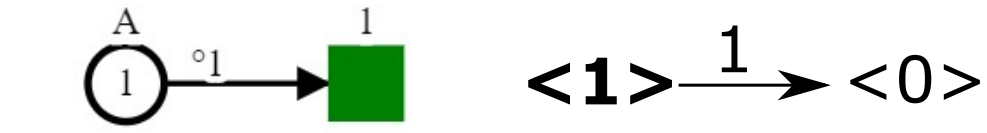
Definice 7 (Slabě živá síť)
Petriho síť je **slabě živá**, pokud pro každý přechod existuje v grafu dosažitelnosti hrana označená tímto přechodem. Pro každý přechod Petriho sítě existuje dosažitelné ohodnocení které povoluje daný přechod.

Definice 8 (Živá síť)
Petriho síť je **živá**, pokud pro každý přechod t a každé ohodnocení M existuje v grafu dosažitelnosti cesta z ohodnocení M do ohodnocení ze kterého vede hrana s označením přechodu t. Pro každý přechod t a každé ohodnocení M existuje dosažitelné ohodnocení M' které přechod t povoluje.

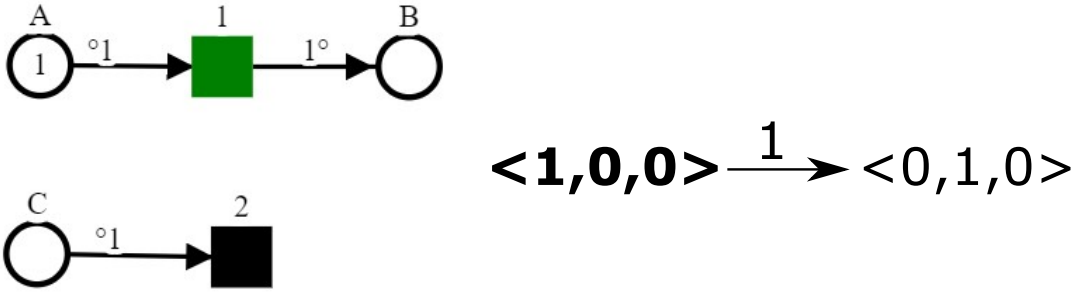
Logickou úvahou a z vlastností grafů pak můžeme určit některé vzájemné závislosti vlastností:

- Síť která je **vratná** nebo/a **živá** je zároveň i **bez mrtvého bodu**.
- Síť která je **bez mrtvého bodu** ne**skončí** a zároveň síť která **skončí** není **bez mrtvého bodu**(pozor, neznamená že síť musí mít alespoň jednu z těchto vlastností).
- Síť která není **slabě živá** nemůže být ani **živá**.

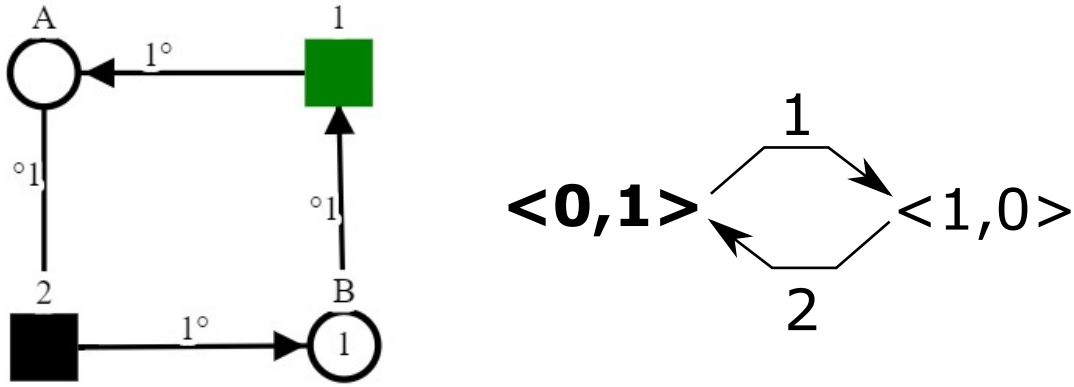
1.5.2 Příklady grafu dosažitelnosti s vlastnostmi



(a)



(b)



(c)

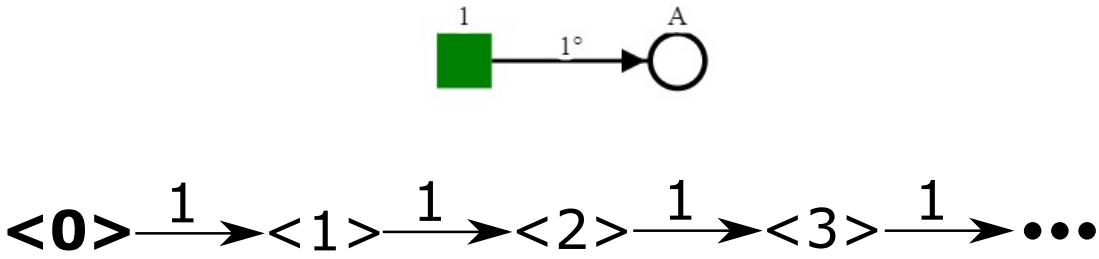
Obrázek 2: Ukázky jednoduchých sítí s grafem dosažitelnosti

- Síť na obrázku 2a **skončí** a je **slabě živá**.
- Síť na obrázku 2b **skončí** a není **slabě živá**.
- Síť na obrázku 2c je **vratná** a **živá**.

Všechny tři sítě jsou ohraničené 1. Také si můžeme všimnout že z výše uvedených vlastností u ukázkových sítí stačí jen vypsané vlastnosti a ostatní se dají odvodit ze závislosti vlastností.

1.6 Graf pokrytí

Hlavní nevýhodou grafu dosažitelnosti je, že může být nekonečný a tudíž je nemožné ho zkonstruovat celý. Můžeme velice jednoduše navrhnout a sestrojit triviální Petriho síť (Obrázek 3) u které by konstrukce jejího grafu dosažitelnosti nikdy neskončila.



Obrázek 3: Příklad neohraničené sítě

Proto existuje upravená verze grafu dosažitelnosti nazvaný graf pokrytí, který může obsahovat tzv. ω ohodnocení, které mimo celých čísel přiřadí alespoň jednomu místu i hodnotu ω symbolizující že místo může nabývat nekonečně vysokého počtu značek. Petriho síť se nemůže nacházet v ω ohodnocení, toto ohodnocení je pouze pro vytvoření abstrakce v grafu pokrytí.

Protože hodnotu ω bereme jako nekonečno pak od ní můžeme odečíst nebo přičíst libovolně velké číslo a hodnota se nezmění.

$$\cdots = \omega - 2 = \omega - 1 = \omega = \omega + 1 = \omega + 2 = \cdots$$

zároveň je taky vyšší jak libovolné přirozené číslo

$$(\forall n \in \mathbb{N}_0) n < \omega$$

Ohodnocení M značíme jako že je ostře menší $<$ než ohodnocení M', pokud pro každé místo p platí $M(p) \leq M'(p)$ a alespoň pro jedno místo p platí $M(p) < M'(p)$.

$$M < M' = ((\forall p \in P) M(p) \leq M'(p)) \wedge (\exists p \in P) M(p) < M'(p)$$

1.6.1 Sestrojení grafu

Sestrojování grafu probíhá postupně přidáváním hran. Nejdříve se přidá počáteční ohodnocení jako kořen grafu. Následně se z grafu vybírají náhodně nevypočítané povolené přechody a pokud vedou do místa, které ještě v grafu není, tak se přidá a pokud je ostře menší než ohodnocení ze kterého je dosažitelné, tak se přidají ω hodnoty na místa ve kterých má více značek. Algoritmus končí výpočet až jsou všechny povolené přechody pro všechny vrcholy v grafu vypočítané.

Sestrojení grafu pokrytí pseudokód 1 MakeCoverabilityGraph.

Algorithm 1 MakeCoverabilityGraph

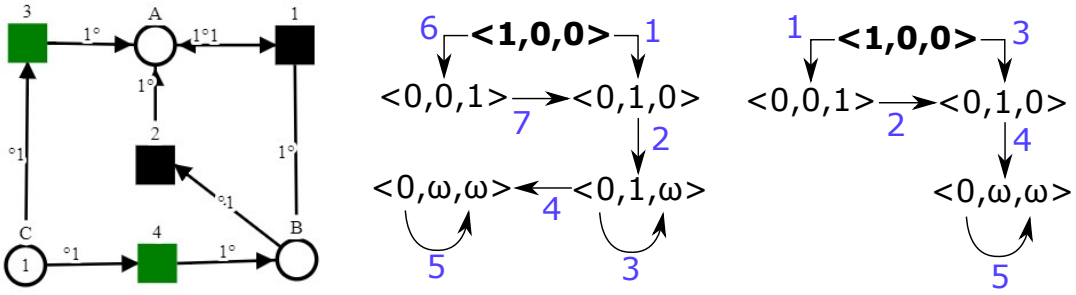
```
1: function MakeCoverabilityGraph(⟨P, T, A, M_0⟩)
2:     ⟨V, E, v_0⟩ := ⟨{M_0}, ∅, M_0⟩
3:     WorkSet := ∅
4:     for all t ∈ EnabledTransitions(P, T, A, M_0) do
5:         WorkSet := WorkSet ∪ {⟨M_0, t⟩}
6:     end for
7:     while WorkSet ≠ ∅ do
8:         ⟨M, t⟩ := RandomElement(WorkSet)
9:         WorkSet := WorkSet \ {⟨M, t⟩}
10:        M' := FireTransition(P, t, A, M)
11:        for all {M'' ∈ V | (M'' →* M ∨ M'' = M) ∧ M'' < M'} do
12:            for all p ∈ P do
13:                if M''(p) < M'(p) then
14:                    M'(p) := ω
15:                end if
16:            end for
17:        end for
18:        if M' ∉ V then
19:            V := V ∪ {M'}
20:            for all t ∈ EnabledTransitions(P, T, A, M') do
21:                WorkSet := WorkSet ∪ {⟨M', t⟩}
22:            end for
23:        end if
24:        E := E ∪ {⟨M, t, M'⟩}
25:    end while
26:    return ⟨V, E, v_0⟩
27: end function
```

Pokud sestrojený graf pokrytí neobsahuje žádné ω ohodnocení, pak je graf pokrytí totožný s grafem dosažitelnosti. Pokud graf pokrytí obsahuje ω ohodnocení, znamená to že graf dosažitelnosti by byl nekonečný a tudíž by nebylo možné ho zkonstruovat celý a nešli by na něm zjišťovat některé nebo všechny vlastnosti. Proto si vystačíme s algoritmem na vytváření grafu pokrytí.

1.6.2 Různé výsledky grafu pokrytí

Při konstrukci grafu pokrytí záleží v jakém pořadí se hrany přidávají a výsledný graf může mít různý počet vrcholů a hran v závislosti na pořadí přidávání hran. V našem algoritmu využíváme funkci *RandomElement*, která vybere náhodný prvek z množiny a snažíme se tak tipovat, jaké pořadí hran bude ideální pro sestrojení nejmenšího grafu. Pokud bychom chtěli sestrojit minimální graf pokrytí, museli bychom nahradit funkci *RandomElement* nějakou funkcí, která by vždy vybrala přechody právě v takovém pořadí, aby došlo k sestrojení minimálního grafu.

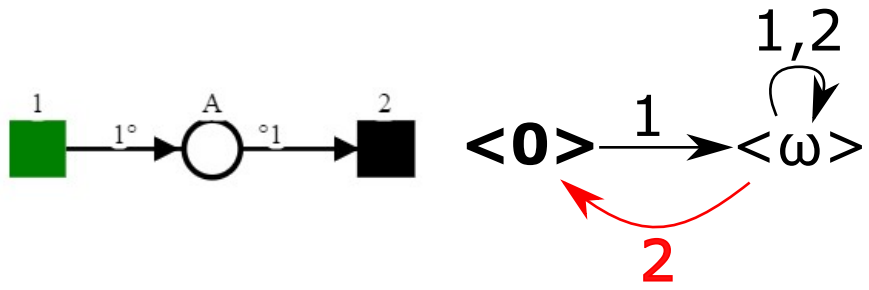
Že záleží na pořadí v jakém se hrany přidávají si můžeme ukázat na síti v obrázku 4. Zde ale musíme dávat pozor, protože v tomto případě čísla neznačí jednotlivé přechody, ale pořadí ve kterém byly hrany přidány.



Obrázek 4: Příklad sítě z knihy Understanding petri nets[1](fig. 14.1) s rozdílnými grafy pokrytí.

1.6.3 Upravená verze vlastností

Oproti grafu dosažitelnosti náš graf pokrytí tak, jak jsme ho sestrojili pomocí algoritmu 1 MakeCoverabilityGraph nemusí obsahovat všechny hrany přechodů, které mohou nastat a to znamená, že v některých případech některé vlastnosti Petriho sítě nejsme schopni určit, protože nám chybí informace o těchto chybějících hranách grafu pokrytí. Problém je částečně způsobený tím, jak máme definovanou hodnotu ω a pokud nějaké místo p je ohodnoceno ω pak už není možné, aby z vrcholu s tímto ohodnocením vedla hrana do vrcholu kde místo p nebude mít hodnotu ω.



Obrázek 5: Příklad neohraničené sítě s chybějící hranou v grafu pokrytí

V Obrázku 5 vidíme červeně zvýrazněnou hranu, která při použití algoritmu 1 MakeCoverabilityGraph chybí. Přitom by tam měla hrana být, protože když budeme neustále opakovat aktivaci přechodu 2, tak se eventuálně (až bude počet aktivací přechodu 2 roven počtu aktivací 1) dostaneme do ohodnocení kde má místo A hodnotu 0, což je zároveň výchozí ohodnocení. Díky této chybějící hraně bychom síť určili jako že není **vratná**, ale přitom ve skutečnosti je. Proto musíme zjistit jestli jsou všechny vlastnosti grafu dosažitelnosti aplikovatlné i na graf pokrytí, případně poupravit nebo rozšířit jejich definici.

Síť je **ohraničená**, pokud graf pokrytí neobsahuje žádné ω ohodnocení

Důkaz (Ohraničenost sítě v grafu pokrytí)
Ohraničenost sítě je určená konečností grafu dosažitelnosti. Nekonečný rozvoj grafu dosažitelnosti je vždy v grafu pokrytí symbolizován ω ohodnoceními. Z toho můžeme vyvodit, že síť je **ohraničená**, pokud její graf pokrytí neobsahuje žádné ω ohodnocení. □

Petriho síť ne**skončí**, pokud graf obsahuje ω ohodnocení.

Důkaz (Konečnost sítě v grafu pokrytí)
Petriho síť ne**skončí**, pokud je její graf dosažitelnosti nekonečný tudíž, stejnou úvahou jako u ohraničenosti můžeme říct, že síť ne**skončí** pokud její graf pokrytí obsahuje ω ohodnocení a skončí, pokud je splněná původní podmínka v definici 4. □

Jestli je síť **vratná** z grafu s ω ohodnocením jsme si už ukázali že díky chybějícím hranám určitelné není. Můžeme si ale jednoduchou úvahou určit množinu případů kdy síť rozhodně vratná není, a to v situaci kdy existuje ω ohodnocení a neexistuje žádný přechod, který by měl větší vstup jak výstup.

Důkaz (Vratnost sítě v grafu pokrytí)
Pokud máme ω ohodnocení tak to mimo jiné znamená že se suma značek všech míst může při opakovaném aktivování některých přechodů zvyšovat neustále. Pak musí existovat i přechod, který tuto sumu snižuje neboli přechod, který má vyšší sumu násobků z místa ${}^{\rightarrow}t$ než do místa $t^{\rightarrow}$. Pokud takový pře-

chod neexistuje přesto že v coverability grafu je ω ohodnocení, pak můžeme s jistotou říct že síť není **vratná** □

Pro určení, jestli je síť **vratná** v ostatních případech bychom potřebovali algoritmus, který vytváří i zpětné hrany z ω ohodnocení.

Petriho síť je **bez mrtvého bodu**, pokud z každého vrcholu grafu pokrytí vede alespoň jedna hrana.

Důkaz (Mrtvý bod v grafu pokrytí)
V tomto případě nemusíme rozlišovat ω ohodnocení a standardní ohodnocení, pokud z něj vede v grafu hrana, znamená to, že v tomto ohodnocení síť neuvázne. □

Petriho síť je **slabě živá**, pokud pro každý přechod existuje v grafu hrana označená tímto přechodem.

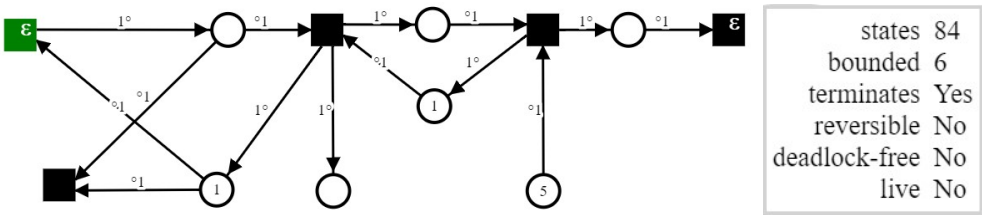
Důkaz (Slabě živá síť v grafu pokrytí)
ω ohodnocení stejně jako v případě určování jestli je síť **bez mrtvého bodu** výsledek neovlivní. Pokud je přechod povolený, nezáleží do jakého hodnocení vede, proto nevadí když chybí jeho hrana do nižšího ohodnocení. □

Petriho síť je **živá**, pokud pro každý přechod t a každé ohodnocení M existuje v grafu cesta z ohodnocení M do ohodnocení ze kterého vede hrana z označením přechodu t.

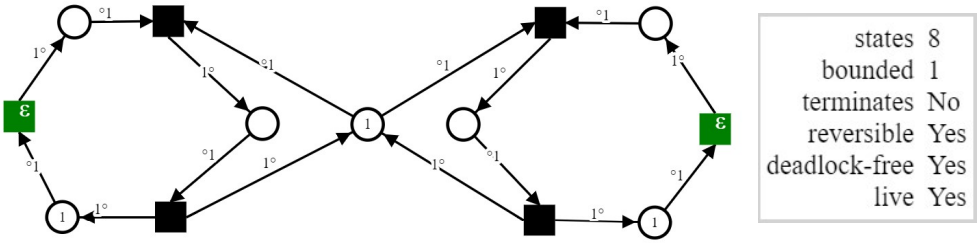
Důkaz (Živá síť v grafu pokrytí)
Když jsou ohodnocení $M < M'$, pak pokud je přechod t povolený v M pak musí být povolený i v M'. Díky tomu můžeme určit že chybějící hrany z ostře větších ohodnocení do menších nejsou potřeba protože by jejich existence stejně neumožňovala přístup k dalším přechodům a proto můžeme živost vyčíst i z grafu pokrytí. □

1.7 Příklady sítí

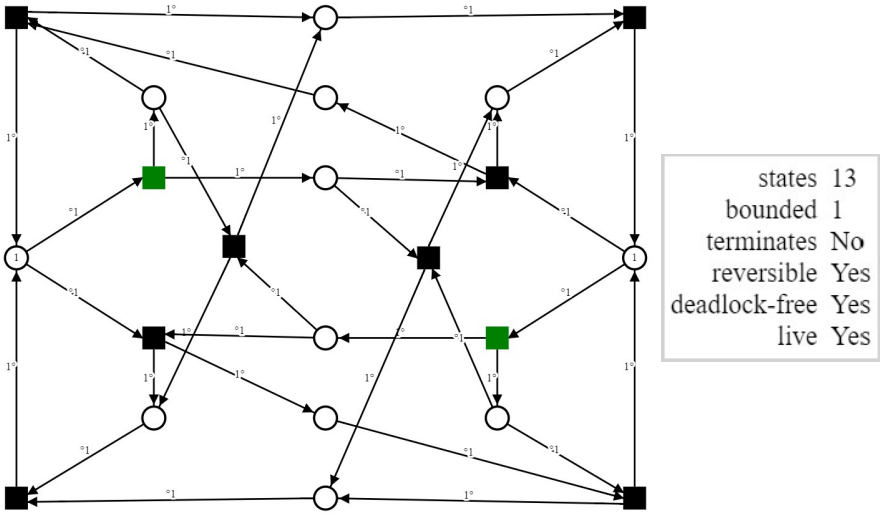
Na obrázcích 6, 7, 8 jsou příklady sítí z knihy Understanding petri nets[1] s výsledky analýz editoru. Odkaz na síť v knice je součástí popisů obrázků.



Obrázek 6: Prodejní automat (fig 3.1)



Obrázek 7: Vzájemné vyloučení (fig 3.2)



Obrázek 8: Crosstalk algoritmus (fig 3.7)

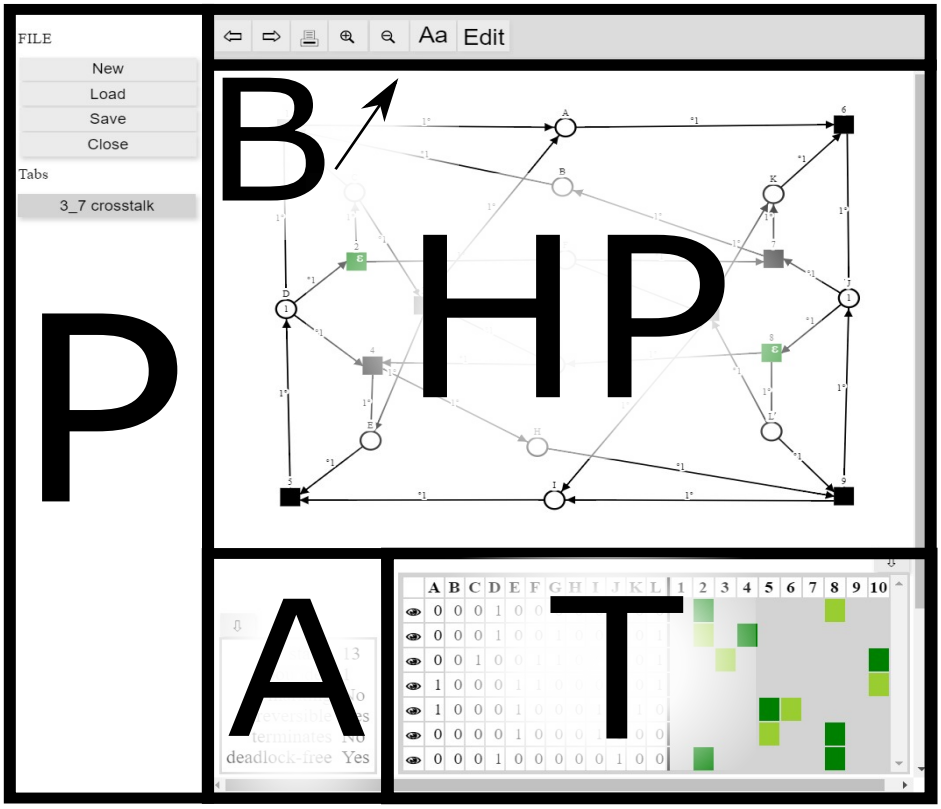
2 Editor

2.1 Systémové požadavky

- Operační systém: Windows 10 (starší verze windows netestovány)
- Ovládání: klávesnice + myš
- Rozlišení obrazovky: minimálně 720p

2.2 Rozložení editoru

Editor je rozložený na několik částí. Všechny tyto části jsou písmeny označené v obrázku 9 Rozložení editoru. Každá část editoru je popsaná ve své vlastní sekci.



Obrázek 9: Rozložení editoru

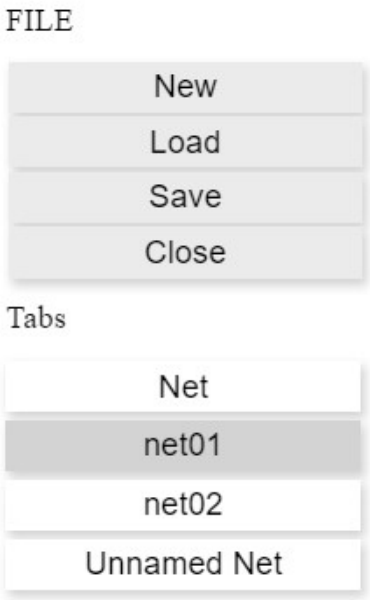
označení	název části editoru	sekce
P	Postranní panel	2.2.1
HP	Hlavní plocha editoru	2.2.2
B	Panel nástrojů editoru	2.2.3
T	Tabulka ohodnocení	2.2.4
A	Výsledky analýzy	2.2.5

2.2.1 Postranní panel

Postranní panel obsahuje tlačítka pro práci se záložkami a samotné záložky. Pod označením **FILE** jsou tlačítka:

New	Vytvoří novou prázdnou záložku.
Load	Otevře dialog pro načtení uložené sítě.
Save	Uloží síť v otevřené záložce. Pokud byla síť již uložena nebo načtena, dialog nabídne uživateli na výběr dvě možnosti. Možnost *yes* uloží a přepíše původní soubor. Možnost *Select file* otevře dialog a uživatel vybere vlastní místo uložení.
Close	Zavře aktuálně otevřenou záložku.

Pod označením **Tabs** jsou pak záložky které se otevírají kliknutím. Záložky jsou nazvané podle jména souboru sítě.



Obrázek 11:
Postranní panel

2.2.2 Ovládání, Hlavní plocha editoru

Editor je navržený tak aby bylo možné jej používat pouze za použití myši bez využití klávesnice. Zároveň oproti ostatním editorům nemá různé nástroje (např. na vkládání různých objektů) a všechny akce editování jsou možné bez toho, aby kurzor opustil hlavní plochu editoru.

Kliknutí levým tlačítkem myši vytvoří přechod. Kliknutím na přechod se začne vytvářet hrana. Pokud se při vytváření hrany klikne do nějakého místa,

tak se na něj hrana připojí. Pokud se klikne do prázdného prostoru, vytvoří se zde nové místo. Vytváření hrany jde zrušit pravým tlačítkem myši.

Při kliknutí na hranu nebo místo se otevře dialog na obrázku 12 editace hodnoty. Při najetí nad editované pole se zobrazí šipky, kterými je možné přidávat nebo ubírat hodnotu, nebo je možné taky do něj kliknout, aby se zde umístil kurzor klávesnice a přitom, když je myš stále nad polem použít kolečko na změnu hodnoty. Dialogy se ukládají kliknutím na OK nebo stisknutím Enter. Kliknutí(levé i pravé) do jiného prostoru hlavní plochy způsobí zavření dialogu bez provedení změny.

Ohodnocení místa se dá měnit i bez otevření dialogu a to najetím myši a rolováním kolečka, zatímco při najetí a rolovaní na hranu pouze prohodí směr hrany. Rolovaní při najetí na přechody pouze odebere nebo přidá přechodu ϵ označení(označení pouze orientační pro uživatele).

Kliknutím a tažením je pak možné jednotlivá místa a přechody přesouvat. Při přesouvání se jednotlivé objekty navzájem odpuzují aby nedošlo k jejich překrytí.

Nakonec pravým tlačítkem je možné místo, přechod nebo hranu odstranit.

Při editaci je možné také využít klávesových zkratek které jsou popsané v sekci 2.2.6.

Pokud uživatel nevyžaduje zobrazené analýzy a tabulku ohodnocení, je možné je tlačítkem nad nimi skrýt.

(a) Editace ohodnocení místa

(b) Editace násobků hrany

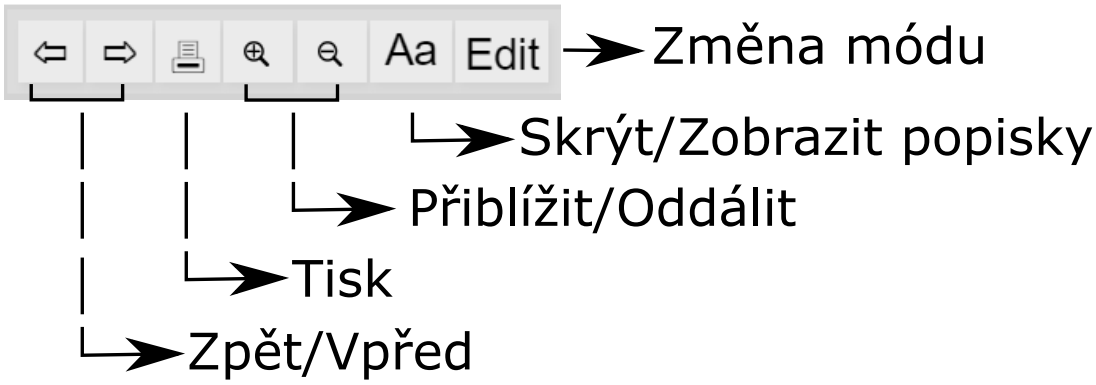
Obrázek 12: Editace hodnot

Pokud je mód nastavený na hodnotu *Run* výše popsané akce myší se vypnou. Jediná akce v tomto módu je aktivace přechodu.

2.2.3 Panel nástrojů editoru

Funkce tlačítek panelu nástrojů zobrazeného v obrázku 13:

Zpět/Vpřed	Vrátí poslední akci nebo zruší vrácení poslední akce.
Tisk	Otevře dialog pro tisk otevřené sítě.
Přiblížit/Oddálit	Přiblíží/Oddálí Petriho síť.
Skrýt/Zobrazit popisky	Skryje nebo zobrazí popisky míst a přechodů.
Změna módu	Přepíná mezi **editovacím módem** a módem **spouštění sítě**.



Obrázek 13: Panel nástrojů editoru

2.2.3.1 Tisk

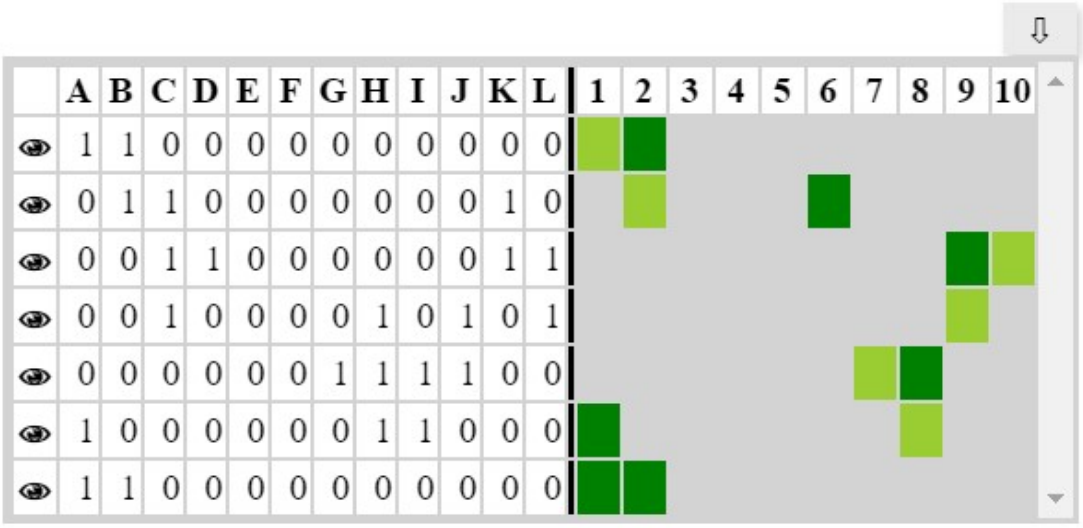
Hlavní plocha má vždy rozměry pro tisk na papír formátu a4. Pro tisk do PDF je potřeba virtuální tiskárna, která tuto funkci umožňuje (např. Microsoft print to PDF).

Tisk nemusí vždy fungovat, je to způsobené chybou ve verzi Electronu která byla k vytvoření programu použita. Pokud tisk nefunguje mělo by stačit editor restartovat.

2.2.4 Tabulka ohodnocení

Tabulka ohodnocení (Obrázek 14) slouží pro rychlé testování sítě uživatelem. Každý řádek v tabulce reprezentuje jedno dosažitelné ohodnocení v levé části tabulky a povolené přechody v tomto ohodnocení jsou zobrazeny zelenými obdelníčky v pravé části tabulky. Uživatel může kdykoliv kliknout na jakýkoliv ze zelených obdelníčků a tím přidá další řádek pod řádek na který kliknul, který bude osahovat nové ohodnocení vzniklé aplikací přechodu který uživatel vybral. Pokud uživatel vybral přechod na řádku pod kterým už řádky jsou, tak budou všechny odebrány a pak až se přidá nový.

Pokud uživatel najede na ikonu oka dojde k zobrazení ohodnocení daného řádku do sítě na hlavní ploše editoru.



	A	B	C	D	E	F	G	H	I	J	K	L	1	2	3	4	5	6	7	8	9	10
	1	1	0	0	0	0	0	0	0	0	0	0										
	0	1	1	0	0	0	0	0	0	0	1	0										
	0	0	1	1	0	0	0	0	0	0	1	1										
	0	0	1	0	0	0	0	1	0	1	0	1										
	0	0	0	0	0	0	1	1	1	1	0	0										
	1	0	0	0	0	0	0	1	1	0	0	0										
	1	1	0	0	0	0	0	0	0	0	0	0										

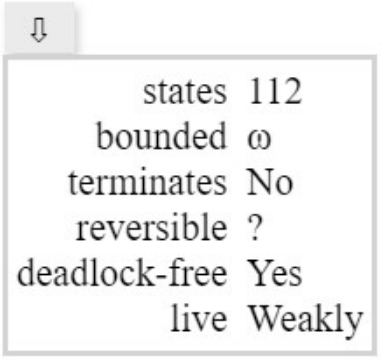
Obrázek 14: Tabulka ohodnocení

2.2.5 Výsledky analýzy

Ve výsledcích analýzy (viz Obrázek 15) jsou zobrazeny vlastnosti sítě podle sekce 1.6.3. Jak jednotlivé vlastnosti přečíst je popsané v tabulce 1.

Název v editoru	Vlastnost	Hodnota	Význam
states	-	číslo	počet stavů sítě
bounded	**ohraničenost**	ω	síť je **neohraničená**
		číslo	určuje **ohraničenost**
terminates	síť **skončí**	*Yes*	síť **skončí**
		No	síť ne**skončí**
reversible	**vratná**	*Yes*	síť je **vratná**
		No	síť není **vratná**
		?	nevíme zda je síť **vratná**
deadlock-free	**mrtvý bodu**	*Yes*	síť je **bez mrtvého bodu**
		No	síť má **mrtvý bod**
live	**živá / slabě živá**	*Yes*	Síť je **živá**
		Weakly	Síť je **slabě živá**
		No	Síť není **slabě živá**

Tabulka 1: Určování vlastností sítě v editoru



Obrázek 15: Ukázka výsledků analýzy sítě

2.2.6 Klávesové zkratky

Seznam klávesových zkratek použitelných v editoru je v tabulce 2.

Zkratka	Akce
Ctrl + Z	Zpět
Ctrl + Shift + Z	Vpřed
Ctrl + O	Otevření uložené sítě
Ctrl + S	Uložení sítě na aktuální záložce
Ctrl + N	Vytvoření nové sítě

Tabulka 2: Klávesové zkratky využitelné v editoru

3 Použité technologie

Následující technologie byly použity pro vytvoření editoru Petriho sítí.

3.1 NodeJS

Technologie která umožňuje využívat jazyk JavaScript pro psaní serverových aplikací. Cílem platformy NodeJS je vytvořit ekosystém pro jednoduší vývoj webových stránek a aplikací, kde stačí pro vytváření funkcionality pouze jeden programovací jazyk.

3.2 Scalable vector graphics (SVG)

SVG je značkovací jazyk vycházející z XML, který popisuje vykreslovaní vektorové grafiky. Je ideální pro vytváření grafiky, kde je vyžadováno aby se dala přibližovat a přitom se neztrácela kvalita obrazu (proto Scalable). V editoru je tato technologie využita v samotnému vykreslovaní sítí.

3.3 TypeScript

TypeScript je opensource programovací jazyk od společnosti Microsoft, který je nádstavbou nad jazykem JavaScript. Jelikož je TypeScript nádstavbou nad programovacím jazykem JavaScript, tak je jakýkoliv validní kód v JavaScriptu validním kódem v TypeScriptu. TypeScript se kompiluje do Javascriptu a proto po stránce funkcionality nenabízí nic navíc, avšak po stránce vývoje nabízí možnost statické typové kontroly se kterou je spjaté fungování našeptávačů v dnešních textových editorech a také nabízí možnost kompilace do starších verzí JavaScript se simulací funkcionality novějších verzí JavaScriptu.

Zdrojový kód 1 je ukázka jednoduchého kódu v TypeScriptu. Můžeme si zde všimnou proměnné *a* do které nakonec nahrajeme řetězec, i přesto že ji máme definovanou jako číslo. Proto si musíme zároveň sami dávat pozor na to jestli tento jazyk využíváme správně a neobcházíme ho, tak že bychom si jím spíš škodili než pomohli.

```
1  // typová inference, a je číslo
2  let a = 10;
3  // řetězec není číslo CHYBA! - nezkompiluje se
4  a = "necislo";
5  // 12 je číslo, validní přiřazení
6  a = 12;
7  // obejití typovosti, proběhne v pořádku ale POZOR
8  // v a už není číslo, ikdyž podle TypeScriptu to tak vypadá
9  (a as any) = "necislo"
10 // definice typu s inicializací, b je číslo nebo text
11 let b: number | string = 10;
12 /**
13  * Funkce fnc je funkce jednoho argumentu který
14  * je složený objekt obsahující
15  * vlasnosti x a y které jsou čísla.
16  * Funkce fnc vrací pravdivostní hodnotu.
17  */
18 function fnc(arg: {x:number,y:number}): boolean {
19    // Kód funkce, pokud všechny větve funkce
20    // nevrací definovanou návratovou hodnotu vyhodí chybu
21 }
```

Zdrojový kód 1: Ukázka TypeScriptu

3.4 Electron

Electron je opensource framework vytvořený v NodeJS, který zaobaluje Windows API (viz zdrojový kód 2) a dohromady se softwarem chromium umožňují vytváření okenních aplikací za pomocí technologií HTML, CSS a JavaScript nebo TypeScript. Electron je stejně jako webová stránka nebo webová aplikace rozdělený na dvě části. „Serverová“ část komunikuje s Windows API a „klientská“ tzv. renderer část se stará už pouze o vykreslování okna. Obě části jsou pak propojené přes *Inter-Process Communication*, což uživatel může využívat přes objekty *ipcRenderer* na straně rendereru a přes *ipcMain* na straně serveru (viz Zdrojový kód 3).

```
1  import { BrowserWindow, Menu, app } from "electron";
2  function createWindow() {
3     // vytvoření samotného okna
4     const mainWindow = new BrowserWindow({
5        width: 400, height: 500,
6        title: 'NazevOkna',
7     });
8     // otevře výchozí stránku
9     mainWindow.loadURL(`file://${__dirname}/index.html`);
10    // odebere horní menu aplikace
11    Menu.setApplicationMenu(null);
12 }
13
14 // počká až vše bude připraveno
15 app.on('ready', createWindow);
```

Zdrojový kód 2: Vytváření okna v elektronu

```
1  //server
2  ipcMain.on("user-event-name", (e, arg)=>{
3    // Do Something
4  })
5
6  //renderer
7  ipcRenderer.send("user-event-name", {/*DATA*/})
```

Zdrojový kód 3: Komunikace v rámci procesu elektronu

3.5 Javascriptová Knihovna Data driven documents (D3)

Knihovna D3 se používá pro zobrazení dat do *document object modelu*. Knihovna obsahuje selektory, které umožňují vybrat zároveň data i DOM objekty a pracovat s nimi najednou. Hlavní výhodou knihovny je její rozdělení selektorů na

Enter, **Exit**, **Update**. Selektor **Enter** se volá, pokud dojde k případu, že má kolekce dat větší počet prvků než je vytvořeno DOM elementů do kterých se mají zobrazovat. **Enter** je tedy selektor který definuje jak se vytváří nové DOM elementy když přibydou data. Naopak **Exit** selektor je pravým opakem **Enter** selektoru a stará se tedy o případy kdy máme méně dat než máme DOM elementů. Poslední selektor je **Update**, který je standardním selektorem (jako je například přímo v JavaScriptu document.querySelector), který přímo aplikuje změny dat do DOM elemntů. Selektory **Enter** a **Exit** mají své příkazy, zatímco selektor **Update** speciální příkaz nemá a je to jen souhrnné označení všech ostatních selektorů. V ukázce (viz Zdrojový kód 4) je jednoduchý kód, který pro každý prvek dat zobrazí jeden řádek s těmito daty.

```
1  import * as d3 from 'd3';
2
3  // vybere v DOM element co má id container
4  const container = d3.select("#container");
5  // data která chceme zobrazit
6  const data = [1, 2, 3, 4];
7  // propojíme data s DOMem
8  container.data(data)
9     // enter selektor
10    .enter()
11    // pro data které nemají element jej přidáme
12    .append("li")
13    // přejde zpět na update selektor
14    .merge(container)
15    // změní text li elementu na číslo z dat
16    .text(d => d)
17    // exit selektor
18    .exit()
19    // odebere přebívající DOM elementy
20    .remove()
21    ;
```

Zdrojový kód 4: Ukázka kódu v knihovně D3

3.6 JavaScript object notation (JSON)

JSON je technologie která využívá notace javascriptových objektů pro ukladání dat. Výhodou tohoto formátu je, že po rozparsování souboru se může s ním hned v javascriptu jednoduše pracovat a také je čitelný pro člověka a je podporovaný ve většině softwaru. Sítě vytvořené v editoru jsou ukládané v tomto formátu. Uložené sítě nejsou určené aby je někdo editoval, ale uložená síť může být třeba použita v jiném programu, který si ji rozparsuje a bude s ní dál pracovat. V ukázce (viz Zdrojový kód 5) je síť tvořená jedním místem s ohodnocením 2, jedním přechodem a hranou směřující do místa.

```
{
  "places": [
    {
      "id": 0,
      "position": {
        "x": 95, "y": 35
      },
      "marking": 2
    }
  ],
  "transitions": [
    {
      "position": {
        "x": 30, "y": 35
      },
      "id": 0,
      "isCold": false
    }
  ],
  "arcs": [
    {
      "place_id": 0,
      "transition_id": 0,
      "toPlace": 1,
      "toTransition": 0
    }
  ]
}
```

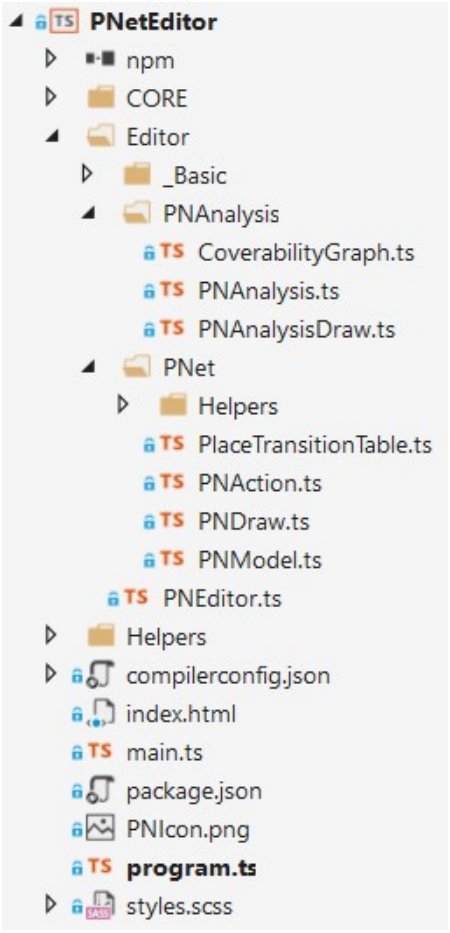
Zdrojový kód 5: Uložení Petriho sítě

4 Stavba programu

Editor je napsán hlavně v renderer straně elektronu. Díky tomu je možné editor v případě potřeby v budoucnu předělat na webovou aplikaci. Struktura programu (Obrázek 16) je tvořená souborem *program.ts*, který má na starosti vytváření hlavního okna a načíst/uložit dialogů. Soubor *main.ts* obsluhuje vytváření a ukládání editorů. Složky CORE a Helpers obsahují univerzální funkcionalitu používanou editorem a složka Editor obsahuje pak samotný editor tvořený třídami Model, Draw a třídami které je propojují do jednoho celku.

4.1 Modely

V programu jsou třídy nazvané model, které obsahují data a starají se o jejich zpracování. Tyto modely obsahují také definice pro převod mezi anonymním objektem a objektem vytvořeným z těchto tříd a díky tomu je jednoduché je serializovat a deserializovat.



Obrázek 16: Struktura programu

4.2 Draw třídy

Draw třídy obsahují logiku k zobrazovaní dat obsažených v modelech a taky zároveň odchytávání akcí způsobených na DOM elementech reprezentujících data v modelech.

4.3 Třída editor

Třída editor propojuje všechny potřebné třídy a zároveň obsahuje obsluhu akcí provedených na třídách draw. Každá síť při načtení nebo vytvoření má vlastní instanci třídy editor.

5 Kompilace

Kompilace programu vyžaduje nainstalovaný *NodeJS*. Pro kompilaci pak stačí pouze spustit skript *src\Build.bat* a program se zkompiluje do složky *src\PNetEditor\dist* kde je jako spustitelný (.exe) soubor.

Pokud dojde ke změnám v TypeScriptových(.ts) souborech, je třeba zkompilovat i je. Ke kompilaci TypeScriptových souborů je potřeba mít visual studio 2017(instalační soubor na CD) s NodeJS sadou nástrojů. Stačí jen otevřít *PNetEditor.sln* a v horní liště visual studia a vybrat možnost *Build→Rebuild solution*.

6 Srovnání s ostatními editory

Ostatní editory nabízí více funkcionality například ve směru analýz, ukládání a tisku. Většina editorů vytváří jednotlivé prvky sítě tak, že na každý prvek existuje nástroj v nástrojové liště. Oproti tomu tento editor nevyžaduje na vytváření sítě žádné přepínání nástrojů. Na základě Fittova zákonu, když není potřeba přejíždět pořád nahorů do lišty pro změnu nástrojů a díky tomu je tento editor rychlejší na vytváření sítí. Tento editor také vyčnívá svou tabulkou ohodnocení, kde si uživatel velice rychle může vyzkoušet a na jednom místě vidět, jestli síť splňuje jeho požadavky.

7 Obsah přiloženého CD

`bin/`
: Adresář obsahuje soubor *pnet_editor.exe* který program spustí.

`doc/`
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu Editor Petriho sítí se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu. Ve složce se nachází soubor řešení Visual Studia 2017 *PNetEditor.sln*, soubor na spouštění kompilace *Build.bat* a složka s kódy a knihovnami.

`readme.txt`
: Tento soubor obsahuje informace pro spuštění a kompilaci bakalářské práce

Navíc CD/DVD obsahuje:

`examples/`
: Obsahuje uložené příklady sítí ze sekce 1.5.2.

install/
: Instalátory aplikací, knihoven potřebných pro kompilaci programu.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Závěr

Při vytváření tohoto editoru jsem se snažil najít něco, čím bude editor vyčnívat od ostatních editorů. Proto jsem se rozhodl zaměřit se na rychlost vytváření sítě za použití pouze myši. Myslím si, že se mi povedlo vytvořit editor, ve kterém se s trochou cviku dá vytvořit libovolná síť velice rychle a pohodlně.

Díky této práci jsem posunul své znalosti dál. Vyzkoušel jsem si různé způsoby vytváření GUI a také se obeznámil s touto problematikou jako celkem. Také jsem rád že jsem díky této práci o trochu obohatil své znalosti na poli teoretické informatiky.

Conclusions

When creating this editor, I tried to find something in which would be editor stand out from other. That's why I decided to focus at the speed of making Petri nets using only mouse. I think that I managed to create an editor in which I can create with a little practice arbitrary network very quickly and conveniently.

Thanks to this work I have moved my knowledge further. I tried different ways to create GUIs and also become familiar with this issue as a whole. I am also glad that I have enriched my knowledge with this work in the field of theoretical computer science.

Literatura

[1] REISIG, Wolfgang. *Understanding petri nets : modeling techniques, analysis methods, case studies*. Berlin: Springer, 2013. ISBN 978-3-642-33277-7.

[2] ***Webová stránka:*** *Data driven documents*. Dostupný z: ⟨https://d3js.org/⟩.

[3] ***Webová stránka:*** *ElectronJS*. Dostupný z: ⟨https://electronjs.org/⟩.

[4] ***Webová stránka:*** *Google-developers JavaScript Promises: an Introduction*. Dostupný z: ⟨https://developers.google.com/web/fundamentals/primers/promises⟩.

[5] ***Webová stránka:*** *Jazyk sass a scss*
. Dostupný z: ⟨https://sass-lang.com/⟩.

[6] ***Webová stránka:*** *NodeJS*. Dostupný z: ⟨https://nodejs.org/⟩.