Simulátor Petriho sítí
PetNetSim

Zpracováno do předmětu
Grafové algoritmy (9GRA)

Ladislav Dobrovský

Fakulta strojního inženýrství

Ústav automatizace a informatiky

2020

Obsah

1 Formální model

Při formálním popisu zanedbávám inhibitory, prioritní, časované a pravděpodobnostní přechody, které si vystačí se slovním popisem funkcionality a příklady.

Graf Petriho sítě

$G = (S, T, W, C)$

S – množina míst

T – množina přechodů

W – množina vah hran W: $(S \times T) \cup (T \times S) \rightarrow \mathbb{N}$

C – kapacity míst C: $S \rightarrow \mathbb{N}$

Petriho síť s počátečním ohodnocením M_0
$N = (S, T, W, C, M_0)$
M_0 – počáteční ohodnocení M_0: $S \rightarrow \mathbb{N}$

Provádění výpočtu Petriho sítě
$\cdot t$ – vstupní místa přechodu t: $\cdot t = \{t \in T, s \in S | W(s,t) > 0\}$
$t \cdot$ – výstupní místa přechodu t: $t \cdot = \{t \in T, s \in S | W(t,s) > 0\}$
E – množina proveditelných přechodů pro ohodnocení M:

$$E = \{t \in T, s \in \cdot t | M(s) \geq W(s,t)\} \cap \{t \in T, s \in t \cdot | M(s) + W(t,s) \leq C(s)\}$$

tj. přechody, které mají ve všech vstupních místech dostatek značek a zároveň dostatečnou kapacitu ve výstupních místech.

A_T – dvojice alternativních přechodů:

$$A_T = \{K = \{t_1, t_2\} | t_1, t_2 \in E \wedge t_1 \neq t_2 \wedge \cdot t_1 \cap \cdot t_2 \neq \emptyset\}$$

A_K – množiny konfliktních přechodů – sjednocení dvojice alternativ s neprázdným průnikem:

$$A_K = \{Q | Q = K_1 \cup ... \cup K_n \wedge K_i \in A_T \wedge K_1 \cap ... \cap K_n = \emptyset\}$$

F – spuštěné přechody

$$F = \{E \setminus \bigcup A_T\} \cup \{\forall Q : \exists! t : Q = \{t\} \wedge Q \subseteq A_K\}$$

tj. jsou spuštěny všechny přechody, které nejsou v alternativě a z konfliktních přechodů se pro každou množinu v A_K vybere právě jeden přechod náhodně.

M_i – ohodnocení v iteraci i, (D_m, D_p – rozdíly ohodnocení); $M_i(s)$ – pro místo s:

$$M_i \overset{F}{\Rightarrow} M_{i+1}$$

$$\text{pro} \quad \begin{array}{l} t \in F \wedge s \in \cdot t \ : \ D_m(s) = \sum W(s,t) \\ t \in F \wedge s \in t \cdot \ : \ D_p(s) = \sum W(t,s) \end{array}$$

$$M_{i+1}(s) = M_i + D_p(s) - D_m(s)$$

Podmínka ukončující výpočet: $F = \emptyset$

2 Implementace

Řešení využívá objektově orientovaného programování, každý objekt může nést jméno, pokud není nastaveno vygeneruje se. Třídy objektů jsou definovány v modulu `petnetsim.elements`:

- `Place` – reprezentuje místo, může mít omezenou kapacitu; neomezená kapacita je definována jako `Place.INF_CAPACITY`
- `Transition` – přechod, dědí z něj třídy:
  - `TransitionPriority` – přechod s prioritou, běžný přechod má nulovou prioritu
  - `TransitionTimed` – časovaný přechod, provádí se pouze, pokud není možné provést běžné přechody. Čas může být konstantní nebo se volí z dodaného rozložení pravděpodobnosti (rovnoměrné, exponenciální, normální, funkce dodaná uživatelem)
  - `TransitionStochastic` – přechod s pravděpodobností, nelze kombinovat s jinými typy přechodů v alternativě
- `Arc` – hrana z místa k přechodu nebo od přechodu do místa,
- `Inhibitor` – inhibiční hrana; rozhodl jsem se nedědit z třídy `Arc`, kde je třeba rozhoduje se přímo podle typu objektu

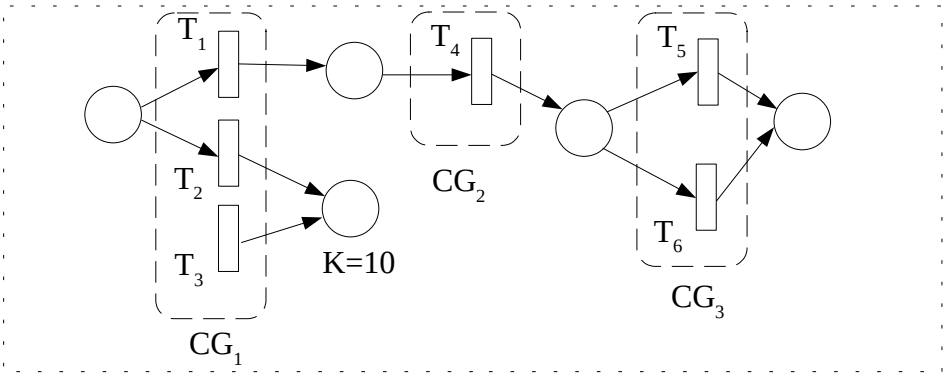
Celou síť reprezentuje objekt třídy `PetriNet`, při konstrukci nastaví konfliktní skupiny a pomocné struktury. Síť po vytvoření nelze měnit. Pokud místo objektů dostane pouze jméno, vytvoří výchozí variantu objektu (místo, přechod, hrana). Kontrolují se duplicity názvů.

2.1 Konfliktní skupiny přechodů (řešení alternativ)

V jednom kroku výpočtu může být paralelně spuštěno více přechodů, pokud jsou povoleny (*enabled*). Pokud přechody sdílejí vstupní (anebo výstupní místa s konečnou kapacitou) jsou v konfliktu a může být spuštěn pouze jeden z každé skupiny (*conflict group, CG*). Protože se graf sítě nemění, je přepočítána maska příslušnosti do skupin (`PetriNet._make_conflict_groups`). Pro každý krok výpočtu je sestaven vektor příznaků spustitelnosti přechodů E_v a ten je pro každou skupinu logicky vynásoben maskou příslušnosti do skupin. Pokud pro skupinu zbyla alespoň jedna logická 1, vybere se náhodně pro spuštění jeden z přechodů.

Poznámka: je možná alternativní implementace, kdy by se ověřovalo zda při sdílení výstupního místa je dostatečná kapacita pro provedení všech povolených přechodů. To však mění počet konfliktních skupin.

Například: T_1 a T_2 sdílejí vstup, T_2 a T_3 sdílejí výstup s konečnou kapacitou, jsou tedy všechny 3 v jedné skupině. T_4 nesdílí vstup ani výstup (to, že výstupní místo pro T_1 je vstupem T_4 nesplňuje podmínku) a tvoří samostatnou skupinu. T_5 a T_6 sdílejí vstup tvoří tak poslední skupinu (sdílený výstup má nekonečnou kapacitu, je tedy nepodstatný).



přechod	T_1	T_2	T_3	T_4	T_5	T_6
povolen E_v	e_1	e_2	e_3	e_4	e_5	e_6
příslušnost CG_1	1	1	1	0	0	0
příslušnost CG_2	0	0	0	1	0	0
příslušnost CG_3	0	0	0	0	1	1
$E_v \wedge CG_1$	e_1	e_2	e_3	0	0	0
$E_v \wedge CG_2$	0	0	0	e_4	0	0
$E_v \wedge CG_3$	0	0	0	0	e_5	e_6

2.2 Inhibitory

Upravují spustitelnost přechodu (viz příklad 3): pokud je ve zdrojovém místě alespoň tolik značek jako váha inhibiční hrany, přechod je neproveditelný. Při provádění přechodu se přes inhibiční hranu neupravuje počet značek ve zdrojovém místě.

2.3 Časované přechody (chybná implementace)

Aktivují se pokud nelze aktivovat v alternativě běžný přechod (viz příklad 6). Pokud dojde k vybrání časovaného přechodu, přejde do stavu čekání i s celou konfliktní skupinou. Přechod je aktivován až v simulaci uplyne zvolený čas čekání (konstantní nebo volen podle definované funkce). Čas simulace se posouvá pokud se v kroku neprovedly žádné nečasované přechody do doby nejkratšího čekajícího přechodu. Časované přechody lze v alternativě kombinovat pouze s běžnými přechody. **Příklad 9 ukazuje na neoptimální kooperaci běžných a časovaných přechodů, je třeba dalšího vývoje programu.**

2.4 Přechody s prioritou

Při alternativě se volí podle nastavené priority přechod s vyšší hodnotou, v případě rovnosti se zvolí náhodně. Prioritní přechody lze v alternativě kombinovat pouze s běžnými přechody, které mají prioritu hodnoty 0.

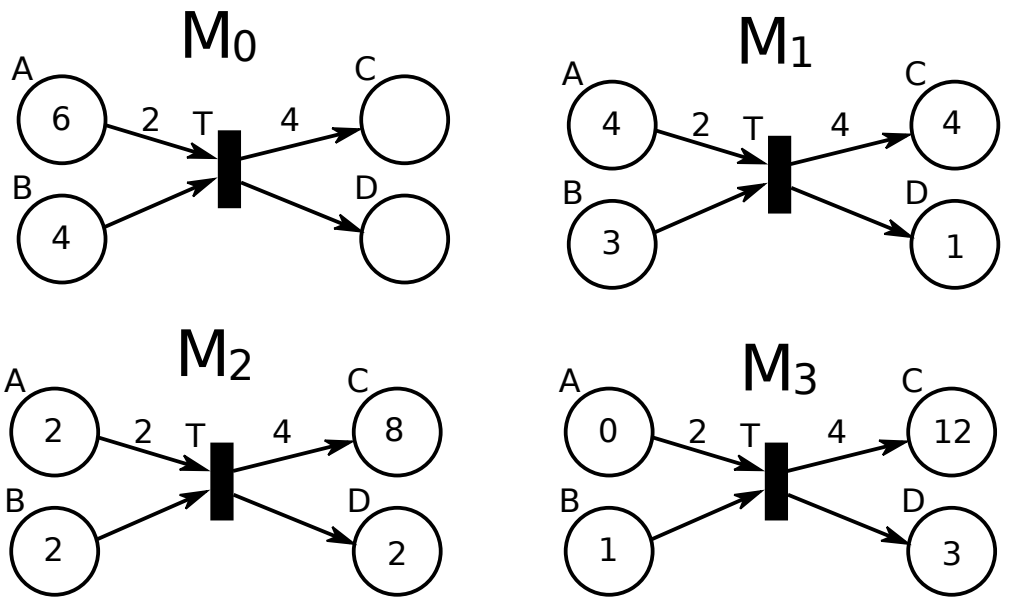
2.5 Přechody s pravděpodobností

Je udaná hodnota pravděpodobnosti spuštění v alternativě. Součet pravděpodobností povolených přechodů v alternativě je váženě normován, aby součet dával 1. Tyto přechody nelze kombinovat v alternativě s jinými typy.

3 Příklady sítí

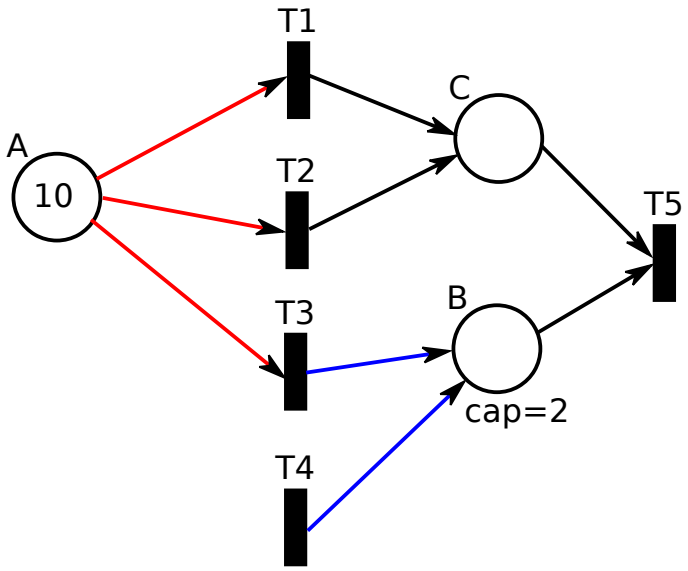
Příklad 1: jednoduchá síť

Síť obsahuje pouze jeden přechod T a 4 místa A, B, C, D. Postup výpočtu je jednoznačný s označeními sítě M_0 až M_3. (sample_001_basic.py)



Příklad 2: alternativa (konfliktní skupiny)

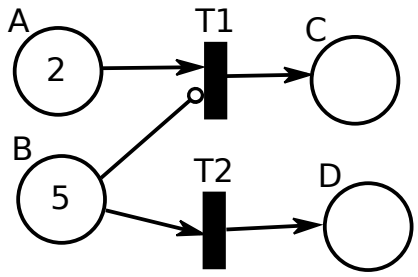
Přechody T1 až T4 tvoří jednu alternativní skupinu spojením červeně a modře označené alternativy. T4 se přidává přes sdílení místa B s přechodem T3 jako výstupu s omezenou kapacitou. Přechody v jedné skupině se spouští náhodně, vždy pouze jeden v kroku výpočtu. T5 se spouští vždy, když jsou splněny podmínky existence značky v místech B i C. (sample_002_conflict_groups.py)



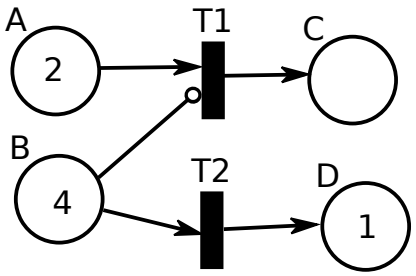
Příklad 3: inhibitor

Hrana z místa B do přechodu T1 je inhibiční, tedy znemožňuje aktivaci T1 dokud je v místě B alespoň jedna značka. Postup výpočtu je jednoznačný s označeními sítě M_0 až M_7 (sample_003_inhibitors.py)

M_0

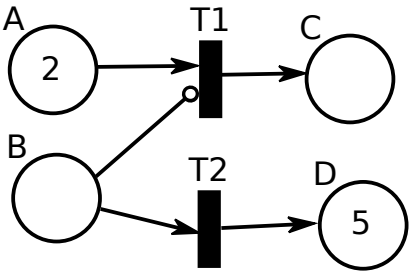


M_1

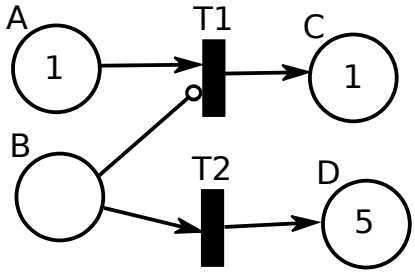


(…)

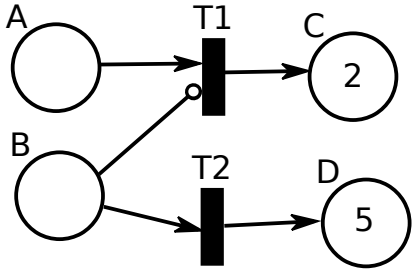
M_5



M_6

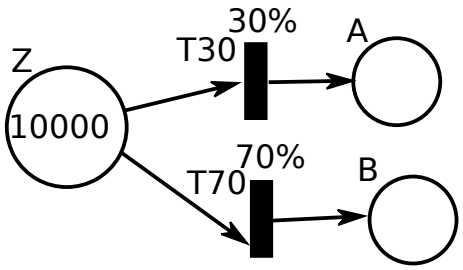


M_7

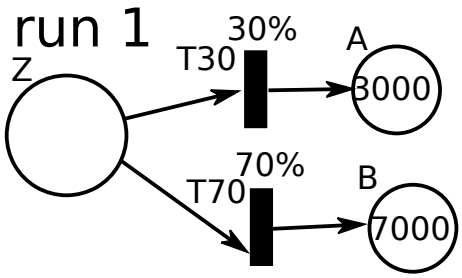


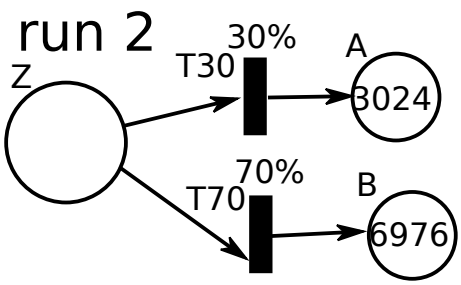
Příklad 4: stochastická alternativa

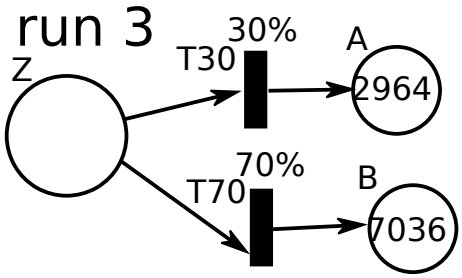
Alternativa přechodů s udáním pravděpodobnosti. (sample_004_stochastic.py)

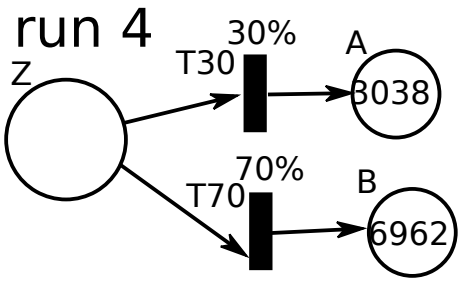


Čtyři běhy simulace sítě dopadly následovně:



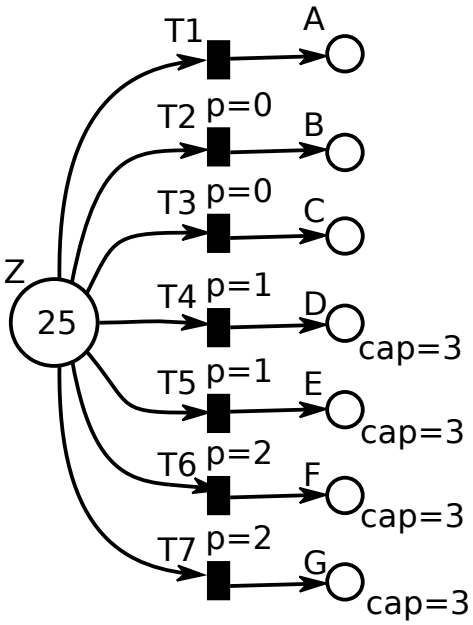






Příklad 5: priority

Přechody s vyššími prioritami se spustí první, alternativa při rovnosti priorit funguje jako normálně. Běžný přechod má prioritu 0. Nejprve se naplní místa F a G, pak D a E, do plné kapacity 3. Místa A, B, C si pak náhodně rozdělí zbytek značek. (sample_005_priority.py)



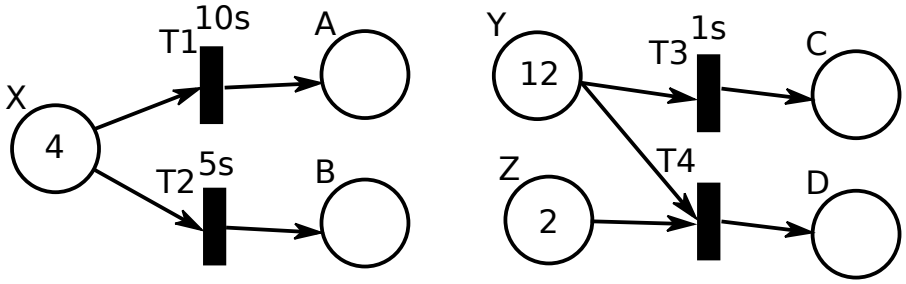
Příklad 6: časované přechody

V alternativě má vždy přednost běžný přechod. Pokud není proveditelný, vybírá se z časovaných. Běžný přechod se provádí okamžitě, neplyne tedy žádný čas simulace, pouze krok výpočtu. Pokud je v alternativě vybrán časovaný přechod, nesmí se jiné přechody v konfliktní skupině spustit dokud není tento přechod proveden, tj. uplyne čas simulace daný přechodem. Přechody v jiných skupinách dále pokračují v krocích výpočtu. K přesunu značek dojde až po uplynutí času.

V síti jsou dva paralelní procesy:

- 4x dojde k alternativě přechodů T1 a T2. V nejkratší variantě 4xT2 celkem 20 sekund, v nejdelší variantě 4xT1 celkem 40 sekund. V průměrné variantě 2xT1 a 2xT2; 30 sekund.
- Nejprve se spustí 2x přechod T4. Pak 10x T3 po sekundě. T3 tedy začne čekat v čase 0, protože přechod T4 neposouvá čas simulace.

(sample_006_timed.py)



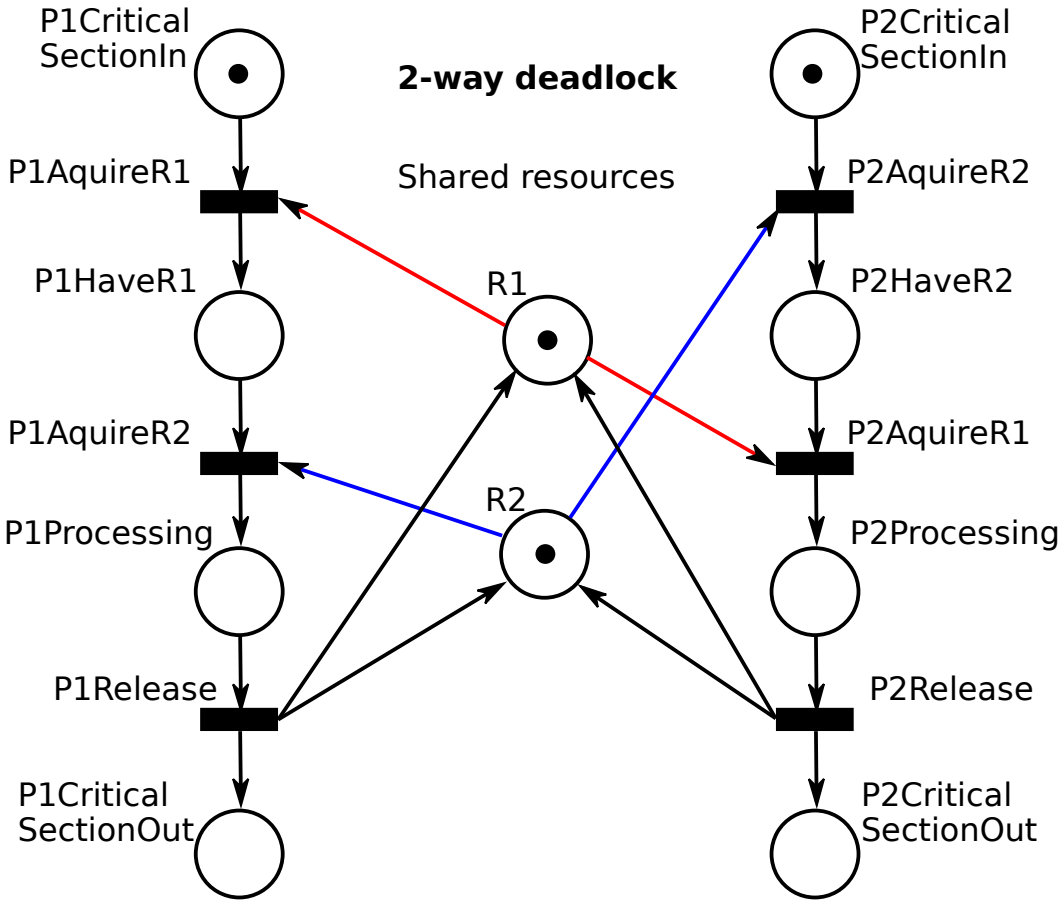
Dva různé průběhy simulace:

step	t [s]	fired
1	0.0	T4
2	0.0	T4
3	1.0	T3
4	2.0	T3
5	3.0	T3
6	4.0	T3
7	5.0	T3
8	6.0	T3
9	7.0	T3
10	8.0	T3
11	9.0	T3
12	10.0	T1, T3
13	20.0	T1
14	25.0	T2
15	30.0	T2
16	30.0	

step	t [s]	fired
1	0.0	T4
2	0.0	T4
3	1.0	T3
4	2.0	T3
5	3.0	T3
6	4.0	T3
7	5.0	T2, T3
8	6.0	T3
9	7.0	T3
10	8.0	T3
11	9.0	T3
12	10.0	T2, T3
13	15.0	T2
14	25.0	T1
15	25.0	

Příklad 7: netriviální uváznutí (deadlock)

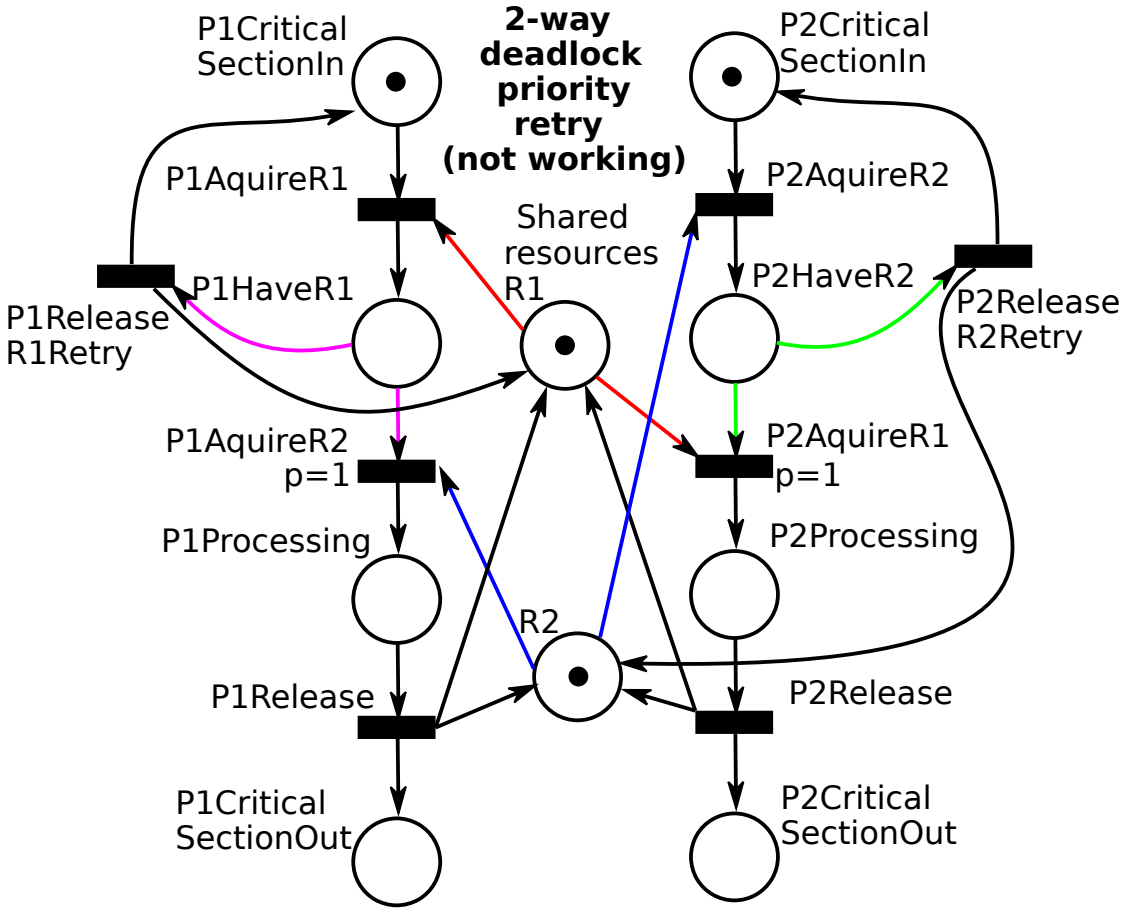
Procesy P1 a P2 uzamykají sdílené zdroje R1 a R2 v opačném pořadí. Nastává uváznutí systému, tj. nejsou žádné spustitelné přechody a zároveň není značka v místech {P1/P2}CriticalSectionOut, které označují konečný stav výpočtu, kdy oba procesy úspěšně dokončili kritickou sekci. Alternativy přechodů jsou označeny barevně (červená a modrá). (*sample_007_deadlock.py*)



Příklad 8: netriviální uváznutí s prioritami (deadlock+bad release)

Situace z příkladu 7 se opakuje s tím, že procesy se pokouší uvolnit první zabraný zdroj pokud se jim nedaří získat druhý a vstup do kritické sekce zopakovat (zaručuje alternativa s prioritou: zelená a magenta). Tato síť modeluje přesně ten případ, kdy jsou procesy v nekonečné smyčce mezi uváznutím a uvolněním zdrojů. Pokud se procesy nemohou domluvit přímo nebo přes arbitra a nečekají jinak dlouho, nemá tato situace řešení.

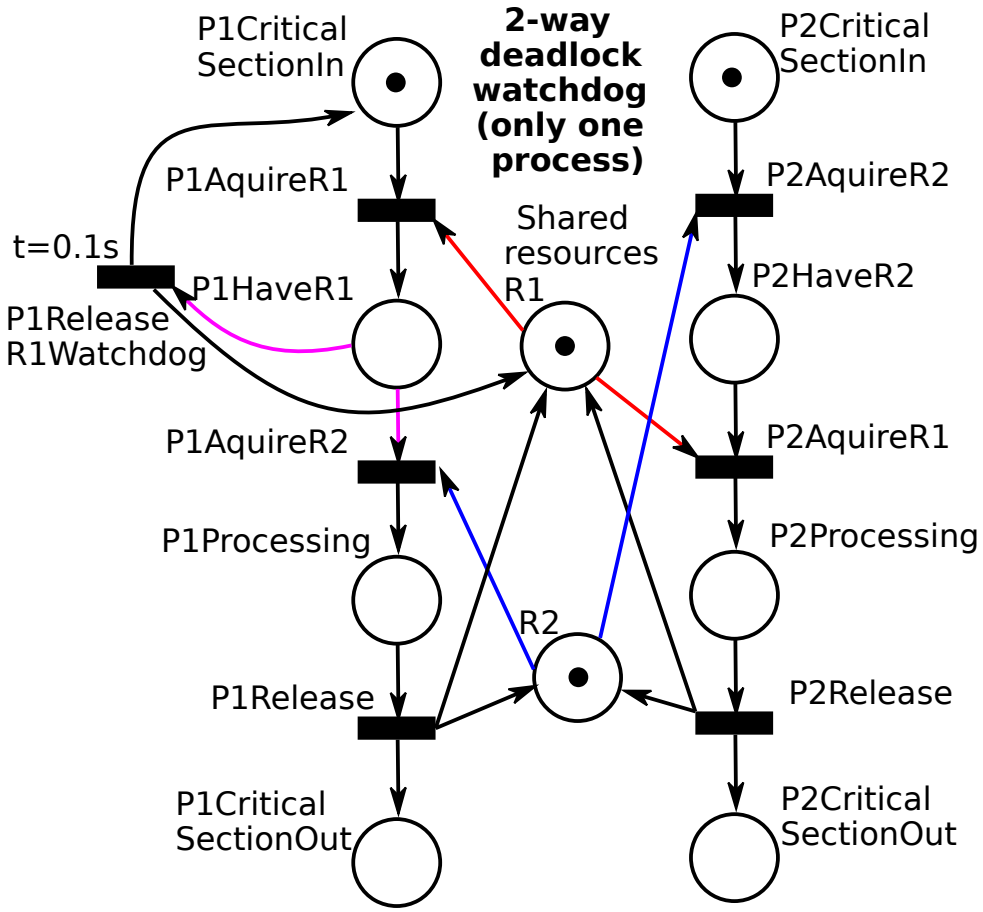
(*sample_008_deadlock_priority.py*)



Příklad 9: netriviální uváznutí s prioritami (deadlock+watchdog)

Situace z příkladů 7 a 8 je řešena přidáním čekání v procesu P1. Pokud by proces P2 vstupoval do kritické sekce opakovaně, může dojít k vyhladovění procesu P1.

Poznámka: příklad ukazuje na neoptimální kooperaci běžných a časovaných přechodů, je třeba dalšího vývoje programu.
(*sample_009_deadlock_P1watchdog.py*)



Příklad 10: test konfliktních skupin

Testování správného vytvoření konfliktních skupin přes výstupní místa s omezenou kapacitou (červená). Základní alternativy jsou barevně označeny: {A, B}, {B, C, D}, {C, D, G}, {E, F}. Ty by se měli sjednocením množin s neprázdným průnikem rozdělit na dvě skupiny: {A, B, C, D, G} a {E, F}.