Študentská vedecká a odborná činnosť sekcia: *Aplikovaná informatika a matematika* ŠVOČ 2009
Fakultné kolo, 29. apríl 2009
Fakulta elektrotechniky a informatiky STU Bratislava

Internetom podporovaná analýza Petriho sietí

Autor: Bc. Zoltán Janík, *zjanik@gmail.com*
Konzultantka: Mgr. Zuzana Ševčíková, KAIVT

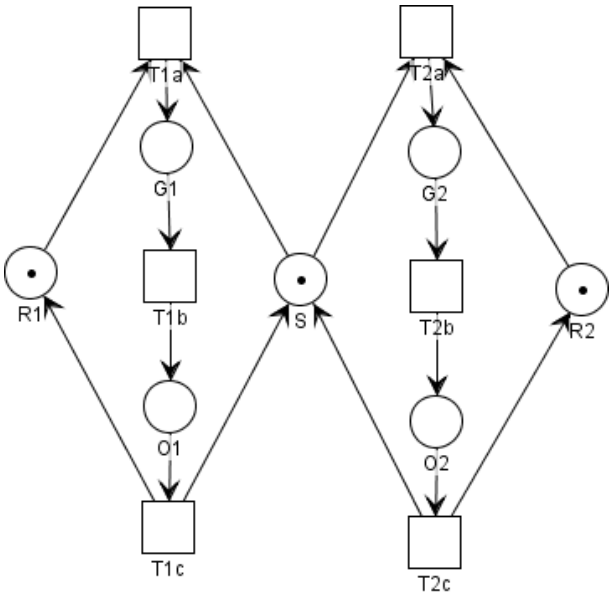
Abstrakt

S Petriho sieťami sa často stretávame pri modelovaní udalostných systémov a workflow procesov. Tento článok v krátkosti oboznámi čitateľa s Petriho sieťami. Jadrom práce je predstavenie a popis internetovej aplikácie naprogramovanej v skriptovacom jazyku PHP, ktorá dokáže spracovať údaje o Petriho sieťach, ktoré sú výstupom desktopovej aplikácie PNEditor. Ako prostriedok analýzy Petriho sietí bol použitý algoritmus, pomocou ktorého sú nájdené minimálne invarianty.

1. Petriho siete

1.1. Úvod

Petriho sieť, resp. place/transition net (skrátene p/t net) je graf s ohodnotenými orientovanými hranami a dvomi typmi uzlov – miesta (places) a prechody (transitions). Platí, že žiadna hrana nemôže spájať dva uzly rovnakého typu. Z toho vyplýva, že hrana nesmie začínať a končiť v tom istom uzle.
Graficky sa miesta najčastejšie označujú kruhmi a prechody obdĺžnikmi (obr. 1).
Hrany sú ohodnotené kladnými celými číslami – multiplicita hrany. Stav miest je daný počtom tokenov v každom mieste (nezáporné číslo). Jednotlivé počty tokenov v miestach sa globálne označujú ako značkovanie v Petriho sieti.
Značkovanie v sieti sa zmení spustením ktoréhokoľvek prechodu v Petriho sieti. Spustenie prechodu spôsobí, že sa počet tokenov zníži v každom mieste, z ktorého smeruje hrana do spusteného prechodu (zmena počtu tokenov je daná multiplicitou hrán). Počet tokenov sa zvýši v miestach, do ktorých smerujú hrany zo spusteného prechodu (takisto podľa multiplicity hrán). Ak treba z miesta odobrať viac tokenov ako je aktuálny počet, takýto prechod potom nie je spustiteľný.
Formálna definícia:
Petriho sieť je štvorica (P,T,F,W), kde P je konečná množina miest a T je konečná množina prechodov.
Platí $P \cap T = \varnothing$, $F \subseteq (P \times T) \cup (T \times P)$, $W : F \to \mathrm{N}^+$. F sa nazýva toková funkcia, W sa nazýva váhová funkcia.



Obr. 1. Model dvoch semaforov

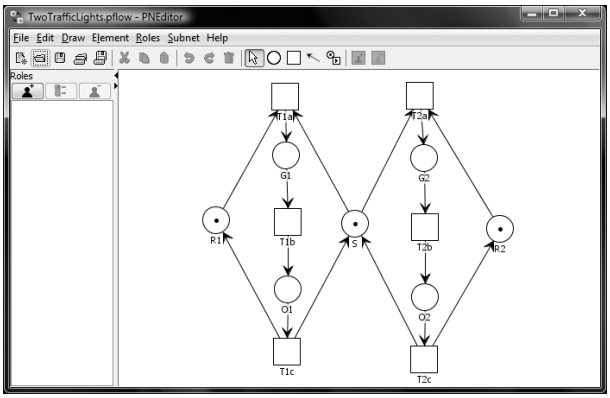
1.2. Invarianty

Invariant je z matematického hľadiska niečo, čo sa nemení vplyvom transformácie, napr. vzdialenosť medzi dvomi číslami na číselnej osi, ktorá sa nemení pričítaním rovnakej hodnoty k obom číslam, pri Petriho sieťach to bude značkovanie siete.
V Petriho sieťach existujú dva typy invariantov:

1. P-invarianty
   - place invariant, vypovedá o stavoch
   - vektor Y, ktorý vynásobením s hocijakým značkovaním dosiahnuteľným z počiatočného značkovania dosiahne ten istý výsledok
   - vektor Y anuluje zľava incidenčnú maticu C, t.j. $Y^T.C = 0^T$
   - nech M_0 je počiatočné značkovanie a M je ľubovoľné značkovanie, vzťah $Y^T.M = Y^T.M_0$ popisuje P-invariant a je označovaný ako zákon zachovania značkovania
2. T-invarianty
   - transition invariant
   - vektor Y stanovujúci počet spustení jednotlivých prechodov, po ktorom bude dosiahnuté pôvodné značkovanie
   - vektor Y anuluje sprava incidenčnú maticu C, t.j. C.Y = 0

2. PNEditor

PNEditor je aplikácia naprogramovaná v Jave 6, ktorá umožňuje vytváranie a úpravy Petriho sietí. Okrem toho je v aplikácii dostupná aj možnosť simulácie vytvorenej siete. V PNEditore je možné namodelovať workflow proces a ten nahrať do webaplikácie PetriFlowWeb.
Vývoj editora je riešený v rámci tímového projektu [2], ktorého riešiteľmi sú Martin Riesz, Dávid Gyurász, Martin Kováč a Martin Sečkár.



Obr. 2. PN Editor

3. Internetová aplikácia

3.1. Objektový model

Transition
–$id –$x –$y –$label
+__construct($id, $x, $y, $label); +__toString(); +getId(); +getX(); +getY(); +getLabel();

Place
–$id –$x –$y –$label –$tokens –$isStatic
+__construct($id, $x, $y, $label, $tokens, $isStatic); +__toString(); +getId(); +getX(); +getY(); +getLabel(); +getTokens(); +showTokens();

Arc
–$sourceId –$destinationId –$multiplicity
+__construct($sourceId, $destinationId, $multiplicity); +__toString(); +getSourceId(); +getDestinationId(); +getMultiplicity(); +showMultiplicity();

Subnet
–$id –$x –$y –$label –$transitions //pole objektov typu Transition –$places //pole objektov typu Place –$arcs //pole objektov typu Arc –$subnets //pole objektov typu Subnet –$debug
+__construct($id, $x, $y, $label); +__toString(); +setDebug($debugMode); +getId(); +getX(); +getY(); +getLabel(); +getTransitions(); +getPlaces(); +getArcs(); +getSubnets(); +getElement($id); +addTransition($transition); +addPlace($place); +addArc($arc); +addSubnet($subnet); +rangeX(); +rangeY(); +minX(); +minY(); +transitionCount(); +placeCount(); +subnetCount(); –getArcsMultiplicityFromSource($sourceId); –getArcsMultiplicityToDestination($destinationId); +getMatrixI(); +getMatrixO(); +getMatrixC(); –getRawC(); +invariants($type, $output);

3.2. Popis metód

3.2.1. Trieda Transition

__construct($id, $x, $y, $label);
Konštruktor, ktorý nastaví všetky členské premenné.
__toString();
Metóda na textový výpis inštancie triedy.

getId(); getX(); getY(); getLabel();
Štandardné gettre, ktorých návratová hodnota je hodnota konkrétnej členskej premennej.

3.2.2. Trieda Place

__construct($id, $x, $y, $label, $tokens, $isStatic);
Konštruktor, ktorý nastaví všetky členské premenné.
__toString();
Metóda na textový výpis inštancie triedy.
getId(); getX(); getY(); getLabel(); getTokens();
Štandardné gettre, ktorých návratová hodnota je hodnota konkrétnej členskej premennej.
showTokens();
Ak je počet tokenov nulový, metóda vráti prázdny reťazec. V opačnom prípade je vrátený počet tokenov. Metóda je využitá napríklad pri generovaní náhľadu, kde nie je žiaduce vypisovať k prechodom nulové tokeny.

3.2.3. Trieda Arc

__construct($id, $x, $y, $label);
Konštruktor, ktorý nastaví niektoré členské premenné. Naplnenie ostatných premenných (polí) je realizované metódami uvedenými nižšie.
__toString();
Metóda na textový výpis inštancie triedy.
getSourceId(); getDestinationId() ;getMultiplicity();
Štandardné gettre, ktorých návratová hodnota je hodnota konkrétnej členskej premennej.
showMultiplicity ();
Ak je multiplicita hrany rovná jednej, metóda vráti prázdny reťazec. V opačnom prípade je vrátená multiplicita. Metóda je využitá napríklad pri generovaní náhľadu, kde nie je žiaduce vypisovať k hranám jednotkové multiplicity.

3.2.4. Trieda Subnet

__construct($sourceId, $destinationId, $multiplicity);
Konštruktor, ktorý nastaví všetky členské premenné.
__toString();
Metóda na textový výpis inštancie triedy.
setDebug($debugMode);
Aktivácia, resp. deaktivácia pomocných výpisov pri výpočtoch.
getId(); getX(); getY(); getLabel(); getTransitions(); getPlaces(); getArcs(); getSubnets();
Štandardné gettre, ktorých návratová hodnota je hodnota konkrétnej členskej premennej.
getElement($id);
Špeciálny getter, ktorý vráti inštanciu takej triedy, ktorá má nastavené zadané ID číslo. ID je unikátne pre všetky miesta, prechody a podsiete.
addTransition($transition); addPlace($place);
addArc($arc); addSubnet($subnet);
Metódy na naplnenie polí. Metódy sú volané postupne počas parsovania súboru .pflow.
rangeX(); rangeY();
Metódy vracajú horizontálny a vertikálny rozsah súradníc jednotlivých uzlov siete. Metódy sú použité pri generovaní náhľadu siete.
minX(); minY();
Metódy vracajú minimálnu horizontálnu a vertikálnu hodnotu súradníc uzlov siete. Metódy sú použité pri generovaní náhľadu siete.
transitionCount(); placeCount();
Počet prechodov a miest v sieti.
getArcsMultiplicityFromSource($sourceId);
Pole hodnôt s multiplicitami všetkých hrán, ktoré smerujú z elementu zadaného ID číslom $sourceId. Táto metóda je privátna, používa sa pri generovaní matice I.
getArcsMultiplicityToDestination($destinationId);
Pole hodnôt s multiplicitami všetkých hrán, ktoré smerujú do elementu zadaného ID číslom $destinationId. Táto metóda je privátna, používa sa pri generovaní matice O.
getMatrixI(); getMatrixO(); getMatrixC();
Výpočet matíc popisujúcich konkrétnu Petriho sieť (asociatívne polia).
getRawC();
Vygenerovanie incidenčnej matice používanej na výpočet invariantov. Matica (pole) má v každom prvku nejakú hodnotu (číslo). Navyše takto vygenerované pole nie je asociatívne. Táto metóda je privátna.
invariants($type, $output);
Výpočet invariantov. Typ invariantov je daný vstupnou premennou $type (place/transition). Výstupná matica môže byť obyčajné alebo asociatívne pole podľa premennej $output (raw/indexed).

3.3. Parsovanie dokumentu .pflow

Modely vytvorené v PNEditore sú uložené v súboroch .pflow vo formáte XML, čo značne zjednodušuje ich použitie. Na parsovanie dokumentov XML má PHP priamu podporu. V tejto aplikácii bolo použité rozšírenie SimpleXML, ktoré skonvertuje textový reťazec (XML dokument) na XML objekt. Každý uzol je potom priamo adresovateľný (napr. `$xml->subnet[0]->id` je ID číslo prvej podsiete). Načítanie samotného obsahu z dokumentu potom prebieha cez cykly prechádzajúce každý uzol XML dokumentu.
Príklad časti skriptu na spracovanie súboru .pflow:

```
$xml = new SimpleXMLElement($text);

$subnet = new Subnet(
      $xml->subnet[0]->id,
      $xml->subnet[0]->x,
      $xml->subnet[0]->y,
      $xml->subnet[0]->label
);

foreach($xml->subnet[0]->place as
$place) {
      $subnet->addPlace(
            new Place(
                  $place->id,
                  $place->x,
                  $place->y,
                  $place->label,
```

```
                $place->tokens,
                $place->isStatic
            )
        );
}

foreach($xml->subnet[0]->transition
as $transition) {
        $subnet->addTransition(
            new Transition(
                $transition->id,
                $transition->x,
                $transition->y,
                $transition->label
            )
        );
}

foreach($xml->subnet[0]->arc as $arc)
{
        $subnet->addArc(
            new Arc(
                $arc->sourceId,
                $arc->destinationId,
                $arc->multiplicity
            )
        );
}
```

3.4. Vykreslenie náhľadu

Náhľad je vygenerovaný prostredníctvom knižnice GD. Skript generujúci náhľad najskôr vytvorí prázdny obrázok typu PNG s rozmermi, ktoré sú získané z metód `rangeX()` a `rangeY()` triedy `Subnet`. Nasleduje vykreslenie všetkých prvkov siete, ktorých umiestnenie je dané členskými premennými `$x`, `$y` v každej z tried (uvedené v kapitole 3.1). Je nutné vykonať korekciu umiestnenia pripočítaním ofsetu (`minX()` a `minY()` z triedy `Subnet`), pretože súradnice v dokumente .pflow nadobúdajú aj záporné hodnoty, a to pri generovaní obrázkov nie je povolené. Jednotlivé elementy sú pospájané šípkami. O to sa postará naprogramovaná funkcia `arrow`, ktorá vypočíta korektný začiatočný a koncový bod spojnice medzi dvomi elementmi uvažujúc to, či ide o miesto alebo prechod (kruh alebo štvorec).
K prvkom sú vypísané doplňujúce informácie: pri hranách sa zobrazí ich multiplicita, ak je väčšia ako 1 (`showMultiplicity()` z triedy `Arc`); pri prechodoch ich popis (`getLabel()` z triedy `Transition`); pri miestach ich popis (`getLabel()` z triedy `Place`) a počet tokenov, ak je väčší ako 0 (`showTokens()` z triedy `Place`).

3.5. Generovanie incidenčnej matice

Incidenčná matica C je jedným z možných spôsobov zápisu Petriho siete. Vznikne z rozdielu matíc $I - O$. Matica I dáva informáciu o multiplicite hrán vstupujúcich do jednotlivých prechodov z každého miesta. Naopak, matica O nesie informáciu o multiplicite hrán vystupujúcich z prechodov a končiacich v miestach.
Príklad matíc pre Petriho sieť z obr. 1 (pre prehľadnosť zobrazené v tabuľkách):

Tab. 1 Matica I

I	T1a	T1b	T1c	T2a	T2b	T2c
G1	0	1	0	0	0	0
O1	0	0	1	0	0	0
R1	1	0	0	0	0	0
G2	0	0	0	0	1	0
O2	0	0	0	0	0	1
R2	0	0	0	1	0	0
S2	1	0	0	1	0	0

Tab. 2 Matica O

O	T1a	T1b	T1c	T2a	T2b	T2c
G1	1	0	0	0	0	0
O1	0	1	0	0	0	0
R1	0	0	1	0	0	0
G2	0	0	0	1	0	0
O2	0	0	0	0	1	0
R2	0	0	0	0	0	1
S2	0	0	1	0	0	1

Tab. 3 Incidenčná matica C

C	T1a	T1b	T1c	T2a	T2b	T2c
G1	-1	1	0	0	0	0
O1	0	-1	1	0	0	0
R1	1	0	-1	0	0	0
G2	0	0	0	-1	1	0
O2	0	0	0	0	-1	1
R2	0	0	0	1	0	-1
S2	1	0	-1	1	0	-1

Vygenerovanie týchto matíc je za pomoci navrhnutého objektového modelu veľmi jednoduché. Pri generovaní matice I sa najskôr vytvorí nulová matica s p riadkami a t stĺpcami, kde p je počet miest a t je počet prechodov. Pre každé miesto v sieti je metódou `getArcsMultiplicityFromSource()` vygenerovaný riadkový vektor, v ktorom sú definované multiplicity hrán smerujúcich z daného miesta do jednotlivých prechodov. Tento vektor je vložený na prislúchajúce miesto v matici I. Matica O je vygenerovaná podobným spôsobom s tým rozdielom, že sú do matice vkladané vektory vygenerované metódou `getArcsMultiplicityToDestination()`.
Takýto vektor obsahuje multiplicity hrán smerujúcich z jednotlivých prechodov do konkrétneho miesta.
Incidenčná matica C je potom rozdielom týchto dvoch matíc. Všetky metódy spomínané v tejto kapitole sú dostupné v triede `Subnet`.

3.6. Výpočet invariantov

V aplikácii bol použitý algoritmus na výpočet minimálnych invariantov (autori Silva, Martinez).

Algoritmus je založený na riadkových operáciách s incidenčnou maticou (kapitola 3.4) rozšírenou o jednotkovú maticu. Jeho výstupom je nájdené riešenie x z rovnice $C^T x = 0$, kde C je incidenčná matica. Premenné p, t reprezentujú počet miest (places) a počet prechodov (transitions) v analyzovanej Petriho sieti. I_p je jednotková matica s rozmermi $p \times p$, I_t je jednotková matica s rozmermi $t \times t$.

Popis algoritmu na hľadanie P-invariantov:

1. A:=**C**;
   B:=$\mathbf{I_p}$
2. cyklus i:=1..**t**
   a. pridať do [A | B] všetky lineárne kombinácie dvojíc vektorov z tejto matice, ktoré majú i-ty prvok nulový
   b. vymazať v [A | B] všetky riadky, ktoré majú i-ty prvok nenulový
   c. vymazať v [A | B] i-ty stĺpec

Na po skončení cyklu bude odstránená celá matica A. Jednotlivé riadky výslednej matice reprezentujú minimálne P-invarianty.

Popis algoritmu na hľadanie T-invariantov:

1. A:=$\mathbf{C^T}$;
   B:=$\mathbf{I_t}$
2. cyklus i:=1..**p**
   a. pridať do [A | B] všetky lineárne kombinácie dvojíc vektorov z tejto matice, ktoré majú i-ty prvok nulový
   b. vymazať v [A | B] všetky riadky, ktoré majú i-ty prvok nenulový
   c. vymazať v [A | B] i-ty stĺpec

Výsledná matica obsahuje minimálne T-invarianty.

Pozn.: v reálnej aplikácii nie sú z matice riadky a stĺpce odstraňované (podľa bodov 2.b. a 2.c.), ale sú len vynulované a je k nim pridaný príznak o vymazaní.

Lineárnou kombináciou vypočítaných invariantov je možné vytvoriť ďalšie invarianty, ktoré už nebudú minimálne.

4. Záver

Implementovaný objektový model použitý v tejto aplikácii bol volený tak, aby sa dal použiť na plnohodnotnú prácu s Petriho sieťami. Inak povedané, tento objektový model úplne popisuje Petriho sieť. Je teda ekvivalentným zápisom k maticovému zápisu (matice I, O) alebo ku grafickému zápisu. Implementované je len nevyhnutné množstvo metód, ktoré sa používajú pri práci s touto sieťou. Vzhľadom k tomu, že sa sieť vždy len rekonštruuje z externého zdroja, v aplikácii nie sú potrebné metódy na zmenu štruktúry siete, pretože je od chvíle načítania siete zo súboru nemenná.
Navrhnutá internetová aplikácia by mala uľahčiť štúdium Petriho sietí. Vygenerované invarianty môžu byť ďalej použité napríklad na analýzu živosti a ohraničenosti siete.
V budúcnosti je naplánovaná integrácia popísanej aplikácie s uvedeným PN Editorom, ktorý bude obsahovať nástroje na analýzu invariantov a ďalších vlastností Petriho sietí. Nástroje analýzy budú takisto integrované aj s internetovou aplikáciou PetriFlowWeb.

5. Odkazy na literatúru a zdroje

[1] JUHÁS, G.: Semantics of Petri Nets: A Comparison, Rigorózna práca, Bratislava 2005, [online] Publikované 30.10.2008 [citované 9.4.2009], Dostupné z http://147.175.106.2/kaivt/Predmety/MSUS/cvicenia?action=AttachFile&do=get&target=PrednaskyText.pdf
[2] Tímový projekt: Generická aplikácia pre manažment workflow procesov, [citované 10.4.2009], Dostupné z http://timak.matmas.net
[3] JANÍK, Z.: Aplikácia na analýzu Petriho sietí, [citované 11.4.2009], Dostupné z http://zolo.yweb.sk/pn/debug.php
[4] HOLEČEK, H: Petriho sítě, Brno 2004, [online] Publikované 8.3.2004 [citované 18.4.2009], Dostupné z http://www.fi.muni.cz/usr/kucera/teaching/pn/pnets.pdf
[5] NAUBER, W: Design and Analysis with Petri Nets (prednášky), [online], Dostupné z http://wwwtcs.inf.tu-dresden.de/~nauber/dapn10.pdf
[6] TAKANO, K: Two efficient methods for computing Petri net invariants, 2001, IEEE