Černobílé a barvené Petriho sítě

poznámky do cvičení z MSI

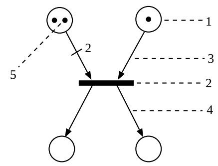
Tomáš Vojnar

1 Motivace

- rozumná grafická reprezentace (ve srovnání s temporální logikou, delšími programy, ...)
- hluboká teoretická báze (různé metody analýzy, ...), jejíž detailní znalost není nutná (podpora nástroji)
- celá řada nástrojů (mnohé jsou free)
- mnoho rozšíření – barvy (datové elementy), hierarchie (makra), OO (metody, třídy), časování, ...

2 Základní pojmy P/T sítí

- Petriho sítě mají charakter bipartitního orientovaného grafu, ve kterém **místa** reprezentují složky možných stavů, **přechody** změny stavů a **hrany** vstupní a výstupní podmínky přechodů.
- příklad 1:



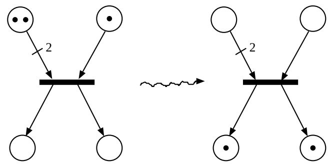
Obrázek 1: Základní elementy P/T sítě: (1) místo, (2) přechod, (3) precondition, (4) postcondition, (5) značení

- Konkrétní stav je určen **značením**, význačnou roli hraje tzv. **počáteční značení**, které reprezentuje počáteční stav systému.
- příklad 2: Příklad 1 můžeme interpretovat tak, že 4 místa reprezentují 4 složky možných stavů (je k dispozici surovina, je k dispozici nástroj, běží opracovávání, nástroj je používán), jediný uvedený přechod reprezentuje jedinou možnou změnu (začátek obrábění). Uvedené značení pak koresponduje s konkrétním stavem (k dispozici 2 ks materiálu a 1 nástroj).

- Hrany mohou mohou mít specifikovánu násobnost.
- Místa mohou mít omezenou kapacitu – lze obejít komplementací.

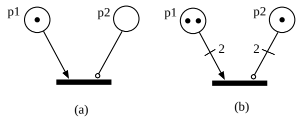
Obrázek 2: Místa s omezenou kapacitou

- Jak již bylo řečeno, ke změně stavu dochází prováděním přechodů. Provedení přechodu sestává z odebrání značení z jeho vstupních míst a umístění značení do míst výstupních, a to na základě vstupních a výstupních hran přechodu.



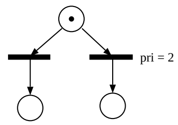
Obrázek 3: Provedení přechodu

- 1. Rozšíření – **inhibiční hrany** – test na 0, respektive na relaci menší než (u obvyklých hran je to větší nebo rovno).



Obrázek 4: Inhibiční hrany: (a) $M(p_1) \geq 1$ *a* $M(p_2) = 0$ (b) $M(p_1) \geq 2$ *a* $M(p_2) < 2$

- 2. Rozšíření – Petriho sítě s **prioritou**. Přechodům přiřadíme prioritu ve formě přirozeného čísla (implicitně 0). Jsou-li proveditelné (až na prioritu) dva přechody současně a jeden má vyšší prioritu (vyšší přiřazenou váhu), provede se tento jako první. Grafická reprezentace – viz obrázek 5.

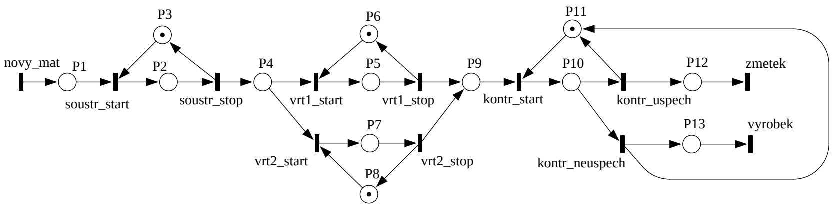


Obrázek 5: Petriho sítě s prioritami

3 Příklady

1. Výrobní systém – přicházející výrobky jsou zpracovány soustruhem, pak jednou ze dvou vrtaček a opouští systém po kontrole jako dobré nebo špatné.

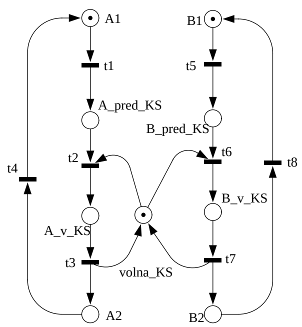
   Řešení viz obrázek 6.



Obrázek 6: Model výrobního systému P/T sítí

   Poznámky: Bylo by dobré omezit počet výrobků v systému, zavést časování pro potřeby analýzy výkonnosti a specifikovat pravděpodopnosti jednotlivých cest. Rovněž vidíme opakování mnoha struktur, což ukazuje na vhodnost nějakého strukturování …

2. Uvažujme dva procesy se sdílenou kritickou sekcí: Proces 1 provede akci A1, potom kritickou sekci KS1, akci A2 a pak se jeho činnost opakuje. Proces 2 provádí cyklicky B1, KS2 a B2. Řešení viz obrázek 7.

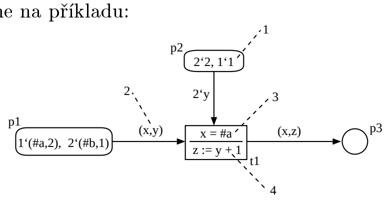


Obrázek 7: Model dvou procesů se sdílenou kritickou sekcí

4 Základy barvených Petriho sítí

- Rozdíl mezi P/T sítěmi a CPN se podobá rozdílu strojového a vyššího jazyka, nicméně P/T sítě je možné (a vhodné) používat častěji než strojový jazyk.
- Základní myšlenkou je individualizace (rozlišení) značek přidáním barev (tj. datových složek) a dále výrazů s těmito hodnotami pracujících.
- Nadále budeme uvažovat CPN zavedené v rámci nástroje PNtalk, existuje však mnoho jiných dialektů. (PNtalk ve skutečnosti pracuje s objektově orientovanými Petriho sítěmi, tuto skutečnost však budeme v dalším ignorovat.)
- PNtalk je nadstavbou Smalltalku a je v něm možné pracovat se všemi objekty Smalltalku. Ty slouží jako datové elementy (barvy), které spojujeme se značkami. Zde je zapotřebí si uvědomit, že ve Smalltalku je (takřka) vše objekt — čísla jsou objekty, řetězce jsou objekty, pole jsou objekty, atd. Proto tedy můžeme (neformálně) říci, že značky v CPN PNtalku jsou obarveny objekty Smalltalku.
- PNtalk (podobně jako Smalltalk) je slabě typovaný, tj. neuvádí se zde typy proměnných, typy míst, apod. Pro konkrétní značku je však vždy jednoznačně dáno, jakého je typu (tj. ze které třídy pochází).
- Základní třídy objektů Smalltalku použitelné v našich CPN:
  - **Čísla**: Je možné pracovat s celými i reálnými čísly, které jsou zapisovány obvyklým způsobem (`1`, `-2`, `3.5e-4`, …) a lze na ně uplatnit běžné operátory (`+`, `-`, `<`, …) a funkce (`sin`, `sqrt`, …).
  - **Symboly**: Jsou podobné řetězcům, zapisují se na počátku se znakem `#` (`#e`). *Poznámka: Symbol `#e` se v PNtalku používá obvykle jako náhrada „černé tečky“.*
  - **Řetězce**: Zapisují se v apostrofech a pro práci s nimi jsou zejména důležité operátory porovnávání (`<`, `>`, …) a konkatenace (`,`).
  - **Znaky**: Zapisují se pomocí konstrukce `$znak`, tedy například `$a`. Nejvýznamnějšími metodami znaků jsou zřejmě operátory porovnávání.
  - **Booleovské konstanty**: Zapisují se pomocí literálů `true` a `false`.
  - **Nedefinovaný objekt**: Zapisuje se pomocí literálu `nil` a hraje významou roli, neboť všechny proměnné, jímž nebyla žádná hodnota dosud přiřazena, mají hodnotu `nil`. Dají se zde užít booleovské metody `isNil` a `notNil`.
  - **Pole**: Vytváří se konstrukcí `#( seznam_prvků )`, například `#( 1 $a #e )`. N-tý prvek se dá zpřístupnit konstrukcí `pole at: n`. *POZOR! Nedoporučuje se používat pole ve stráži přechodů, neboť to vede k problémům s testováním jejich proveditelnosti …*
- Kromě základních tříd Smalltalku je ovšem možné vytvářet vlastní (např. prioritní frontu apod.), to však vyžaduje znalost Smalltalku.
- Vlastní CPN sestává ze **sítě** reprezentující strukturu modelu a **inskripcí** (výrazů). Je vždy nutno pečlivě zvážit, které aspekty modelu budou vyjářeny Petriho sítí (typicky řídící struktura) a které inskripcemi (typicky manipulace s daty).

- Notaci CPN uvedeme na příkladu:



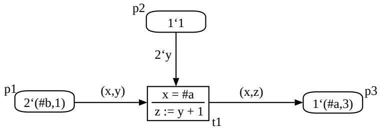
Obrázek 8: Základní elementy CPN používaných v PNtalku

Vysvětlení: **Počáteční značení** (1) má podobu multimnožiny zapsané následujícím způsobem: $n_1‘c_1, n_2‘c_2, \ldots$, kde n_i je celé číslo a c_i literál nebo n-tice. **Hranový výraz** (2) říká, co se odebere, resp. umístí, z, resp. do, příslušného místa. Hranové výrazy mají charakter specifikace multimnožin $n_1‘c_1, n_2‘c_2, \ldots$. Kde n_i je buďto celočíselný literál (jedničku lze vynechat), nebo proměnná, a c_i je literál, proměnná, nebo n-tice. Jestliže hranový výraz na vstupní hraně přechodu obsahuje proměnné, je nutné před provedením tohoto přechodu specifikovat jejich **navázání**, tj. přiřazení hodnot těmto proměnným tak, aby bylo možné přechod provést, tj. aby byla splněna podmínka ve **stráži přechodu** (3) a ve všech vstupních místech byl dostatek značek určených navázáním. Navázání proměnných na výstupních hranách je pak odvozeno ze vstupního navázání a z přiřazení provedených v **akci přechodu** (4).

Poznámka 1: Prázdný hranový výraz odpovídá `#e`, tj. přemístění „černé tečky“.

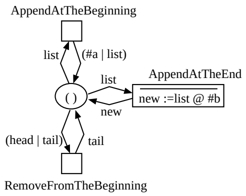
Poznámka 2: Je možné použít tzv. **testovací** hranu, která se zobrazuje se šipkami na obou koncích. Tato hrana značky neodebírá, pouze vyžaduje před provedením přechodu přítomnost určitého značení v určitém místě.

Příklad provedení přechodu: Přechod `t1` je možné provést pro navázání `x = #a, y = 2` – viz obrázek 9.



Obrázek 9: CPN z obrázku 8 po provedení přechodu `t1`

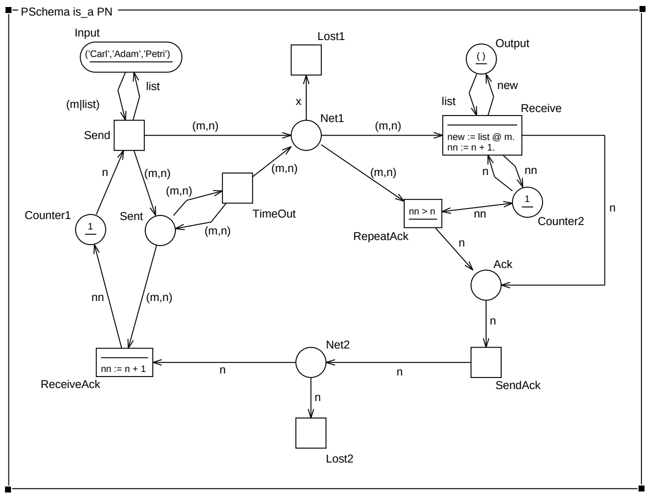
- Kromě n-tic je možno v PNtalku pracovat také se **seznamy inspirovanými Prologem** – viz obrázek 10:



Obrázek 10: Práce se seznamy v PNtalku

5 Příklad

- Jednoduchý model 2 komunikujících uzlů, kdy máme zajistit spolehlivý přenos množiny zpráv z uzlu 1 do uzlu 2 (tj. uspořádání na vstupu do kanálu stejné jako na výstupu, nic se netratí, nic nevzniká navíc).
- Použijeme P-schéma potvrzování: Odešleme očíslovanou zprávu. Na druhé straně se zpráva se správným číslem přijme (jiné se ignorují) a pošle se potvrzení. Přijde-li potvrzení, lze vyslat další zprávu. Jinak se po vypršení time-outu opakuje vysílání předchozí zprávy.
- Řešení viz obrázek 11:



Obrázek 11: Model komunikace s potvrzováním dle P-schématu

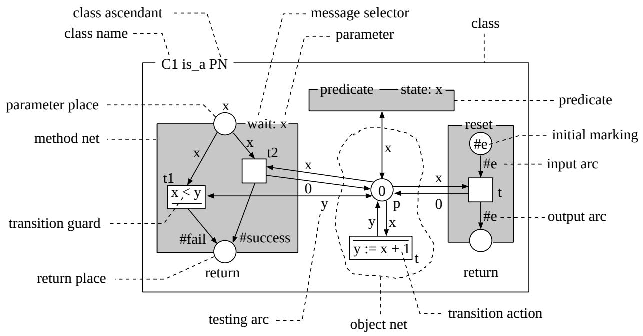
6 Objektově orientované Petriho sítě

Pro zájemce uvedeme v této sekci kopii stručného neformálního popisu **objektové orientace v PNtalku** převzatou z některého z článků na tuto tématiku:

> The OOPN formalism is characterized by a Smalltalk-based object-orientation enriched with concurrency and polymorphic transition execution, which allow for message sending, waiting for and accepting responses, creating new objects, and performing primitive computations.
>
> OOPNs are based on viewing objects as active servers which offer reentrant services to other objects. Services provided by objects, as well as independent activities of objects, are described by Petri nets — services by *method nets*, object activities by *object nets*. Tokens in nets represent references to objects.
>
> An OOPN consists of Petri nets organized in classes (for an example see figure 12). Every class consists of an object net describing the internal activity of objects of this

class and a set of dynamically instantiable method nets describing how these objects respond to messages. All method nets of a given class share access to the appropriate object net (places of the object net are accessible for transitions of method nets). Each method net has parameter places and a return place. Class inheritance is defined by the inheritance of object nets[1], together with sets of method nets[2]. Classes can also contain predicate methods which allow for atomic testing their objects' states without any side-effects.

Every object is either trivial (e.g. a number or a string) or it is an instance of some Petri net-described class consisting of one instance of the appropriate object net and several currently running instances of method nets. When an object receives a message, a new instance of the corresponding method net is created, parameters are put into the parameter places and the instance of the method net is being executed concurrently with all other net instances until the return place receives a token. Then the value of the token in the return place is passed to the message sender as the result of the requested service, and the instance of the method net is deleted. Message sending and object creations are specified as actions attached to transitions. Execution of transitions is polymorphic — invoked methods depend on classes of message receivers which are unknown at the compile time.



Obrázek 12: A simple class demonstrating the notion of the OOPN formalism

7 Odkazy

Systém PNtalk a více informací o něm mohou zájemci získat na URL:
`http://www.fee.vutbr.cz/~janousek/pntalk/pntalk.html`

[1]Inherited transitions and places identified by their names can be redefined and new places or transitions can be added.
[2]The implementation of inherited methods can be changed and new methods can be added.