**TECHNICKÁ UNIVERZITA V KOŠICIACH**
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**
**KATEDRA POČÍTAČOV A INFORMATIKY**

# Doplnkové učebné texty z Formálnych špecifikácií systémov (predbežné)

***Ing. Štefan Korečko, PhD.***

*Tento dokument štrukturálne a obsahovo vychádza z prednášok predmetu "Formálne špecifikácie systémov"* ***prof. Ing. Štefana Hudáka, DrSc.*** *a je primárme určený pre študentov popri zamestnaní v tomto predmete.*

**Košice, 2005-2008**

Posledná modifikácia: 19. septembra 2009

# Obsah

# Kapitola 1

# Úvod

*Formálne špecifikácie*, resp. *FDT (Formal Description Techniques)*, resp. *Formálne metódy* sú metódy stojace na pevných matematických základoch, umožňujúce jednoznačný popis systémov. Ich hlavným cieľom je vyjadriť formálnym spôsobom požiadavky na navrhovaný systém tak, aby bolo možné analyzovať a verifikovať funkciu systému.

Motivácie pre použitie formálnych špecifikácií:

- neustále narastajúca veľkosť a zložitosť používaných diskrétnych systémov,
- narastajúca závislosť ľudskej spoločnosti na počítačových systémoch,
- potreba zvládnutia chýb a zlého správania sa systémov systematickým spôsobom pomocou automatizovaného prístupu.

**Definícia 1.0.1.** Diskrétny systém *je dvojica* $S = (C, B)$*, kde* $B$ *je správanie systému a* $C$ *je štruktúra systému.*

Jedna štruktúra môže vykazovať viac správaní a pre jedno správanie môže existojať viac štruktúr. *Správanie* môže byť *sekvenčné* alebo *nesekvenčné*, nesekvenčné správanie môže byť *paralelné* alebo *súbežné*(concurrent).

Pod pojmom *návrh systému* rozumieme určenie $C$ zo známeho $B$ a *analýza systému* je určenie $B$ na základe známej $C$.

Klasifikácia (rozdelenie) formálnych špecifikácií (FDT):

- *FDT orientované na popis správania sa systému:* konečné automaty, Petriho siete, CSP - komunikujúce sekvenčné procesy, procesné algebry.

- *FDT orientované/založené na údajové/ých štruktúry/ach:* ADT - abstraktné údajové typy, algebraické špecifikácie, objektovo-orientované programovacie jazyky.
- *Funkcionálne orientované FDT:* funkcie, booleovské funkcie, funkcionálne programovacie jazyky.
- *FDT orientované na vlastnosti:* logiky (propozičná, prvého rádu, temporálne, ...).

Prekážky na ceste použitia FDT:

1. potreba zvládnutia formálneho aparátu týchto metód,
2. nedostatok praktických skúseností s použitím niektorých FDT pre rozsiahle systémy,
3. široká priepasť medzi formálnou špecifikáciou a konečnou podobou systému (napr. spustiteľným kódom),
4. potreba zvládnutia reálneho času,
5. potreba zvládnutia nesekvenčného správania (napr. súbežnosť),
6. potreba zvládnutia de/kompozičného prístupu k návrhu a analýze,
7. absencia dobrého teoretického základu niektorých FDT pre použitie v reálnom procese návrhu a analýzy.

Doterajšie skúsenosti ukazujú, že neexistuje a pravdepodobne ani nebude existovať jediná univerzálna formálna metóda. Preto pre popis, analýzu, verifikáciu rôznych aspektov diskrétnych systémov potrebujeme rôzne formálne metódy.

Súčasný vývoj v oblasti formálnych metód sa uberá týmito smermi:

1. kombinovanie existujúcich FDT,
2. rozširovanie existujúcich FDT (napr. rozšírenia o možnosť nakladania s reálnym časom),
3. zúženie priepasti medzi formálnou špecifikáciou a konečnou podobou systému (spustiteľným kódom, ...).

# Kapitola 2

# Petriho siete

Petriho siete patria medzi formálne metódy orientované na popis správania sa systémov. Sú formálnym a grafickým jazykom, vhodným pre modelovanie systémov s paralelizmom [36]. Ich autorom je C.A. Petri, ktorý jazyk Petriho sietí formálne definoval v dizertačnej práci 'Kommunikation mit Automaten' [25]. Sú určené na modelovanie systémov s vlastnosťami, ako paralelizmus, synchronizácia, distribuované spracovanie alebo zdieľanie zdrojov (podrobnejšie v [27]). Veľmi cenné sú ich analytické vlastnosti, umožňujúce automatické odvodenie invariantných vlastností popisovaného systému, čo v iných formálnych špecifikáciách možné nie je [15]. Boli s úspechom použité ako model pre množstvo aplikácií, ako paralelné systémy, komunikačné protokoly, vyhodnocovanie výkonnosti a systémy odolné voči poruchám.

Petirho siete sú

- model vypočítateľnosti s vyjadrovacou silou na vyjadrenie vlastností vychádzajúcich za rámec modelov vypočítateľnosti založených na sekvenčnom a deterministickom princípe,
- formálny prostriedok špecifikácie systémov určitej triedy a reprezentácie systémov vykazujúcich paralelné, asynchrónne a súbežné správanie.
- formálna špecifikácia s jedinečnými analytickými vlastnosťami, umožňujúcimi automaticky odvodiť invariantné vlastnosti systému.

## 2.1 Základné pojmy

Obyčajná Petriho sieť (PS) je definovaná ako štvorica [12]:

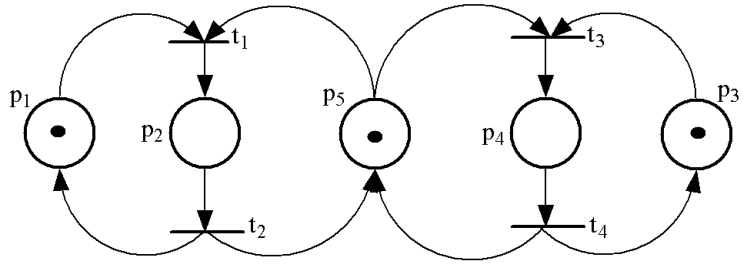
**Definícia 2.1.1.** *Obyčajná Petriho sieť $N$ je usporiadaná štvorica*

$$N = (P, T, pre, post)$$

*kde* *P je konečná množina miest (Places),*
*T je konečná množina prechodov (Transitions),*
$pre : P \times T \rightarrow \{0, 1\}$ *je pre-podmienka,*
$post : P \times T \rightarrow \{0, 1\}$ *je post-podmienka*

□

Hodnoty predikátov *pre* a *post* sa obyčajne zapisujú v tabuľkách. Miesta sa tiež zvyknú označovať ako podmienky (conditions) a prechody ako udalosti (events). Petriho siete sú veľmi často reprezentované orientovaným bipartitným grafom s dvoma druhmi vrcholov - miestami a prechodmi. Z obrázku




P = {p1,p2,p3,p4,p5}
T = {t1,t2,t3,t4}

| *P* | *T* | *pre(p,t)* |
|---|---|---|
| p1 | t1 | 1 |
| p5 | t1 | 1 |
| p2 | t2 | 1 |
| p5 | t3 | 1 |
| p3 | t3 | 1 |
| p4 | t4 | 1 |
| inak | | 0 |

| *P* | *T* | *post(p,t)* |
|---|---|---|
| p1 | t2 | 1 |
| p2 | t1 | 1 |
| p5 | t2 | 1 |
| p3 | t4 | 1 |
| p4 | t3 | 1 |
| p5 | t4 | 1 |
| inak | | 0 |

Obr. 2.1: Petriho sieť - graf, množiny P a T, tabuľky funkcií pre a post.

2.1, kde je znázornená Petriho sieť modelujúca vzájomné vylúčenie dvoch procesov, je jasný vzťah medzi definíciou funkcií *pre*, *post* a štruktúrou Petriho siete: ak $pre(p, t) = 1$, potom miesto $p$ je pre-miestom prechodu $t$, čo sa v grafe znázorní orientovanou hranou z $p$ do $t$. Podobne ak $post(p, t) = 1$, potom miesto $p$ je post-miestom prechodu $t$, čo sa v grafe znázorní orientovanou hranou z $t$ do $p$.

Ďalej je možné definovať množiny:

$${}^\bullet t = \{p | pre(p, t) \neq 0\}$$ množina pre-podmienok $t$

$$t^\bullet = \{p | post(p,t) \neq 0\} \text{ množina post-podmienok } t$$

$$p^\bullet = \{t | pre(p,t) \neq 0\} \text{ množina post-prechodov } p$$

$$^\bullet p = \{t | post(p,t) \neq 0\} \text{ množina pre-prechodov } p$$

*Konfiguráciu* (stav) Petriho siete $N = (P, T, pre, post)$ vyjadruje jej *značenie* $m$ (marking) , ktoré je definované ako funkcia z množiny miest $P$ do množiny prirodzených čísel $\mathbb{N}$:

$$m : P \to \mathbb{N}$$

Hodnota $m(p)$ vyjadruje počet značiek (tokenov) v mieste $p$ a takisto platnosť podmienky $p$: hovoríme, že *podmienka p platí*, ak $m(p) \neq 0$. Značenie Petriho siete možno zapísať aj vo vektorovej podobe, ako

$$\vec{m} = (m(p_1), m(p_2), \ldots, m(p_{|P|}))$$

Vektorovú podobu značenia, $\vec{m}$, zvyčajne zapisujeme bez šípky, teda ako $m$. Väčšinou je PS definovaná spolu s počiatočným značením $m_0$ ako *inicializovaná* PS $N_0 = (N, m_0) = (P, S, pre, post, m_0)$.

### 2.1.1 Výpočet Petriho siete

*Zmena stavu* Petriho siete nastane *vykonaním* (odpálením, realizáciou) niektorého *vykonateľného* (realizovateľného, prípustného) prechodu:

**Definícia 2.1.2.** *Prechod $t \in T$ v PS $N = (P, T, pre, post)$ je* vykonateľný *v značení $m \in \mathbb{N}^{|P|}$ (zapisujeme $m \overset{t}{\vdash}$), ak je splnená podmienka:*

$$\forall p(p \in {}^\bullet t) : m(p) \geq pre(p,t)$$

Výsledkom vykonania prechodu *je nové značenie $m' \in \mathbb{N}^{|P|}$ (zapisujeme $m \overset{t}{\vdash} m'$):*

$$m'(p) = \begin{cases} m(p) - pre(p,t) & ak\ p \in^\bullet t \wedge p \notin t^\bullet \\ m(p) + post(p,t) & ak\ p \in t^\bullet \wedge p \notin^\bullet t \\ m(p) - pre(p,t) + post(p,t) & ak\ p \in^\bullet t \cap t^\bullet \\ m(p) & inak \end{cases}$$

□

Ak značenie PS umožňuje realizáciu viacerých prechodov naraz, vykoná sa iba jeden - ľubovoľný. To je dané vlastnosťou nedeterminizmu PS.

Sekvencia $\sigma \in T^*$, $\sigma = t_1, t_2 ... t_r$ je *prípustná sekvencia prechodov* PS $N$ (zapisujeme $m \overset{\sigma}{\vdash}$), ak existujú značenia $m_0, m_1, ... m_r$ také, že $m_{i-1} \overset{t_i}{\vdash} m_i, i = 1, 2...r$.
Značenie $m_r$ sa nazýva *dosiahnuteľné značenie z* $m_0$ *cez* $\sigma$ (zapisujeme $m \overset{\sigma}{\vdash} m_r$).

Pre danú PS $N = (P, T, pre, post, m_0)$ je definovaná množina dosiahnuteľných značení ($*$ predstavuje ľubovoľnú sekvenciu prechodov):

$$\mathcal{R}(N_0) = \{m | m_0 \overset{*}{\vdash} m\}$$

*Jazyk* PS $N_0$, $N_0 = (N, m_0) = (P, S, pre, post, m_0)$ je množina

$$\mathcal{L}(N_0) = \{\sigma \in T^* | m_0 \overset{\sigma}{\vdash}\}$$

Vzťah medzi jazykom PS a formálnymi jazykmi je popísaný v [12].

### 2.1.2 Živosť Petriho siete

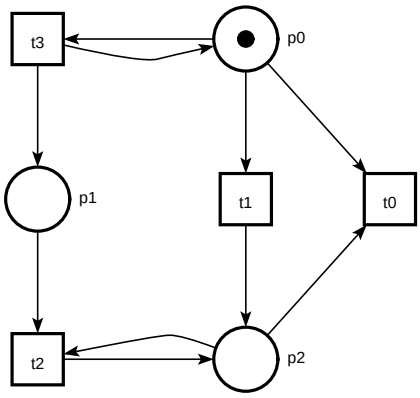
**Definícia 2.1.3.** *Majme PS* $N_0 = (P, T, pre, post, m_0)$ *a prechod* $t \in T$. *Prechod* $t$ *sa nazýva*

- 0- živý $(L_0(t)) \Leftrightarrow_{def} \neg(\exists \sigma \in T^* : m_0 \overset{\sigma}{\vdash} \wedge \sigma(t) \geq 1)$
- 1- živý $(L_1(t)) \Leftrightarrow_{def} \exists\, \sigma \in T^* : m_0 \overset{\sigma}{\vdash} \wedge \sigma(t) \geq 1$
- 2- živý $(L_2(t)) \Leftrightarrow_{def} pre\ \forall r \in \mathbb{N}\ \exists \sigma \in T^* : m_0 \overset{\sigma}{\vdash} \wedge \sigma(t) \geq r$
- 3- živý $(L_3(t)) \Leftrightarrow_{def} \exists\, \sigma \in T^* : m_0 \overset{\sigma}{\vdash} \wedge \sigma(t) \geq \omega$[1]
- 4- živý $(L_4(t)) \Leftrightarrow_{def} \forall m \in \mathcal{R}(N_0) : \exists\, \sigma \in T^* : m \overset{\sigma}{\vdash} \wedge \sigma(t) \geq 1$[2]

Zápis $\sigma(t)$ je početnosť výskytu prechodu $t$ v sekvencii $\sigma$. a $\omega$ predstavuje číslo väčšie ako ktorékoľvek prirodzené číslo. Jednotlivé typy živosti prechodov ilustruje sieť na obrázku 2.2, prevzatá z [23].

Živost Petriho siete definujeme na základe živosti jej prechodov:

[1] tzn. $t$ sa v $\sigma$ vyskytuje neobmedzene veľa krát.
[2] tzn. $t$ je 1-živý v každom značení siete.




Obr. 2.2: Petriho sieť, v ktorej je prechod $t_0$ 0-živý, $t_1$ 1-živý, $t_2$ 2-živý a $t_3$ 3-živý.

**Definícia 2.1.4.** *Petriho sieť $N_0$ je* i-živá *práve vtedy, ak každý jej prechod je i-živý:*

$$L_i(N_0) \Leftrightarrow_{def} \forall\, t \in T : L_i(t)$$

*Petriho sieť $N_0$ sa nazýva* živá *práve vtedy, ak je 4-živá:*

$$Live(N_0) \Leftrightarrow_{def} L_4(N_0)$$

## 2.2 Algoritmické problémy Petriho sietí

Pre petriho siete je možné formulovať viacero problémov. Najdôležitejšie z nich sú uvedené v tejto časti. Predpokladáme, že PS $N_0 = (P, T, pre, post, m_0)$ a $k = |P|$, $n = |T|$.

1. **Problém dosiahnuteľnosti** (*RP-Reachability Problem*) **.**

   Majme PS $N_0$ a $k$-rozmerný nezáporný celočíselný vektor $q$. Problém či $q \in R(N_0)$ je nazývaný *problémom dosiahnuteľnosti* (*inštanciou problému dosiahnuteľnosti*) Petriho siete $N_0$ (pre stav $q$).

   O tomto probléme bližšie pojednáva kapitola 2.8.

2. **Problém živosti** (*LP-Liveness Problem*) **.**

   Problém rozhodnúť, či PS $N_0$ je $i$-živá, $i = 0, 1, 2, 3, 4$ sa nazýva *problém i-živosti* siete $N_0$.

   Problém rozhodnúť, či PS $N_0$ je živá sa nazýva *problém živosti* siete $N_0$.

3. **Problém ohraničenosti** (*BP-Boundedness Problem*) **.**

   Problém rozhodnúť, či PS $N_0$ je ohraničená (bounded) sa nazýva *problém ohraničenosti* siete $N_0$.

   PS $N_0$ je *ohraničená*, ak $\exists r \in \mathbb{N}$ také, že pre $\forall m \in R(N_0)$ a pre $\forall p \in P$ platí, že $m(p) \leq r$

4. **Problém pokrytia** (*CP-Coverability Problem*) **.**

   Majme PS $N_0$ a $q \in \mathbb{N}^k$. *Problém pokrytia*, $Cow(N_0, q)$, je zistiť, či $\exists m \in R(N_0) : q \leq m$

   Hovoríme, že $q \leq m \Leftrightarrow \forall i (1 \leq i \leq k) : q_i \leq m_i$.

5. **Problém reverzibility** (*RvP-Reversibility Problem*) **.**

   Problém rozhodnúť, či PS $N_0$ je reverzibilná (reversible) sa nazýva *problém reverzibility* siete $N_0$.

   PS $N_0$ je *reverzibilná*, ak pre $\forall m \in R(N_0) : \ \exists \sigma \in T^* : (m \overset{\sigma}{\vdash} m_0)$

6. **Problém domovského stavu** (*HSP-Home State Problem*) **.**

   Problém rozhodnúť, či PS $N = (P, T, pre, post)$ má domovský stav (home state) sa nazýva *problém domovského stavu* siete $N$.

   Značenie $m \in R(N)$ je domovský stav (home state marking) ak pre $\forall m' \in R(N) : \ \exists \sigma \in T^* : (m' \overset{\sigma}{\vdash} m)$. Značenie $m$ je teda dosiahnuteľné z ľubovoľného značenia siete.

   RvP je vlastne obmedzenou verziou HSP.

7. **Problém ekvivalencie a obsiahnuteľnosti** (*Equality (EqP) and Containment (CtP) Problem*) **.**

   Majme 2 Petriho siete $N_{0i} = (P_i, T_i, pre_i, post_i, m_{0i}), \ i = 1, 2$, také že $|P_1| = |P_2|$.

   Problém rozhodnúť, či sú tieto siete ekvivalentné ($N_{01} \approx N_{02}$), tzn. či $R(N_{01}) = R(N_{02})$ sa nazýva *problém ekvivalencie.*

   Problém rozhodnúť, či $N_{01}$ je obsiahnutá v $N_{02}$ ($N_{01}$ $leqN_{02}$), tzn. či $R(N_{01}) \subseteq R(N_{02})$ sa nazýva *problém obsiahnuteľnosti.*

8. **Deadlock problém** (*Deadlock Problem*) **.**

   Problém Deadlock($N_0$) je určiť, či existuje $m \in R(N_0)$, ktoré je deadlockové.

   Značenie $m \in R(N_0$, je *deadlockové*, ak neexistuje $t \in T$, ktorý je realizovateľný v $m$.

## 2.3 Analytické vlastnosti Petriho sietí

Petriho siete disponujú jedinečnými analytickými vlastnosťami, umožňujúcimi automatické odvodenie *invariantných vlastností* siete. Invariantné vlastnosti systému sú také, ktoré platia vždy, v každom stave systému. Pre Petriho sieť vieme odvodiť dva typy invariantov a to *S-invarianty*, invarianty miest a *T-invarianty*, invarianty prechodov.

Predtým, ako zadefinujeme samotné invarianty, uvedieme definíciu *incidenčnej matice*, ktorá je ďalšou reprezentáciu Petriho siete a *Parikhovho obrazu*, ktorý vyjadruje početnosť jednotlivých prechodov v danej sekvencii prechodov.

**Definícia 2.3.1.** *Incidenčná matica Petriho siete $N$ je matica*

$$c = \{c_{ij}\},\ c_{ij} = post(p_i, t_j) - pre(p_i, t_j)$$

□

Reprezentácia incidenčnou maticou je jednoznačná len pre tzv. *čisté siete* (pure nets)[3].

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
|---|---|---|---|---|
| $p_1$ | $-1$ | 1 | 0 | 0 |
| $p_2$ | 1 | $-1$ | 0 | 0 |
| $p_3$ | 0 | 0 | $-1$ | 1 |
| $p_4$ | 0 | 0 | 1 | $-1$ |
| $p_5$ | $-1$ | 1 | $-1$ | 1 |

Obr. 2.3: Incidenčná matica siete z obrázka 2.1.

**Definícia 2.3.2.** *Majme PS $N_0 = (P, T, pre, post, m_0)$ a prípustnú sekvenciu prechodov $\sigma \in T^*$.* Parikhov obraz $\sigma$ *vzhľadom na $T = \{t_1, \ldots, t_n\}$ je vektor*

$$\Psi(\sigma) = (\sigma(t_1), \ldots, \sigma(t_n))$$

*, kde $\sigma(t_i)$ je početnosť výskytu prechodu $t_i$ v $\sigma$.*

□

Pomocou incidenčnej matice a Parikhovho obrazu možno vypočítať výsledné značenie po vykonaní ľubovoľnej prípustnej sekvencie prechodov $\sigma$:

[3]Čisté siete neobsahujú vlastné slučky (self loops):$\neg\exists(t \in T \wedge p \in P) : pre(p,t) \neq 0 \wedge post(p,t) \neq 0$

**Veta 2.3.1.** *Majme PS* $N_0 = (P, T, pre, post, m_0)$ *a reťazec* $\sigma \in T^*$. *Ak* $m_0 \overset{\sigma}{\vdash} m$ *tak* $m = m_0 + (c.(\Psi(\sigma))^T)^T$.

□

Definície 2.3.3 a 2.3.4 definujú S a T invarianty a vety 2.3.2 a 2.3.3 objasňujú ich význam.

**Definícia 2.3.3.** *Majme PS* $N_0 = (P, T, pre, post, m_0)$ *a k-zložkový stĺpcový vektor* $X$, $k = |P|$. *Ak platí, že* $c^T.X = 0$, *potom* $X$ *je* S-invariant *siete* $N_0$.

□

**Veta 2.3.2.** *Majme stĺpcový vektor* $X$, *ktorý je S-invariant siete* $N_0$. *Potom pre* $\forall m\,(m \in R(N_0)) :\ m.X = m_0.X$.

□

**Definícia 2.3.4.** *Majme PS* $N_0 = (P, T, pre, post, m_0)$ *a n-zložkový stĺpcový vektor* $Y$, $n = |T|$. *Ak platí, že* $c.Y = 0$, *potom* $Y$ *je* T-invariant *siete* $N_0$.

□

**Veta 2.3.3.** *Majme stĺpcový vektor* $Y$, *ktorý je T-invariant siete* $N_0$. *Majme reťazec* $\sigma \in T^*$ *taký že* $m_0 \overset{\sigma}{\vdash} m$ *a Parikhov obraz* $\sigma$ *je* $Y$ *(*$\Psi(\sigma) = Y$*). Potom* $m_0 = m$.

□

## 2.4 Základné modifikácie Petriho sietí

V tejto časti uvedieme dve základné modifikácie obyčajných Petriho sietí a to *zovšeobecnené Petriho siete* (Generalised Petri nets, GPN) a *kapacitne obmedzené Petriho siete* (Capacity bounded Petri nets, CBPN) . V oboch prípadoch ide o technické modifikácie, keďže každú GPN aj CBPN je možné jednoducho previesť na ekvivalentnú obyčajnú PS.

V GPN sú prípustné viacnásobné hrany, keďže hodnoty *pre* a *post* môžu byť väčšie ako 1. Pre výpočet GPN platí definícia 2.1.2.

**Definícia 2.4.1.** *Zovšeobecnená Petriho sieť* $N$ *je usporiadaná pätica*

$$N_0 = (P, T, pre, post, m_0)$$

*kde* $P$ *je konečná množina miest,*
$T$ *je konečná množina prechodov,*
$pre : P \times T \to \mathbb{N}$ *je pre-podmienka,*
$post : P \times T \to \mathbb{N}$ *je post-podmienka,*
$m_0$, $m_0 \in \mathbb{N}^{|P|}$, *je počiatočné značenie*

□

V CBPN je pre každé miesto $p$ stanovená jeho kapacita $K(p)$, určujúca maximálnu hodnotu značenia (tzn. maximálny počet tokenov) v mieste $p$.

**Definícia 2.4.2.** *Kapacitne obmedzená Petriho sieť $N$ je usporiadaná šestica*

$$N_0 = (P, T, pre, post, K, m_0)$$

*kde* $P$, $T$, $pre$, $post$, $m_0$ *sú definované, ako pre obyčajnú Petriho sieť,* $K : P \to \mathbb{N} - \{0\}$ *je funkcia definujúca kapacitu miest.*

□

Kvôli obmedzeniu kapacitou je vykonateľnosť pre CBPN definovaná mierne odlišne, ako pre obyčajné PS, no výpočet nového značenia ostáva rovnaký:

**Definícia 2.4.3.** *Prechod $t \in T$ v CBPN $N_0 = (P, T, pre, post, K, m_0)$ je vykonateľný v značení $\vec{m} \in \mathbb{N}^{|P|}$, ak je splnená podmienka:*

$$\forall p\,(p \in {}^\bullet t) : m(p) \geq pre(p,t) \wedge \forall p'\,(p' \in t^\bullet) : m(p') \leq K(p') - post(p',t)$$

*Výsledkom vykonania prechodu je nové značenie, definované rovnako, ako v definícii 2.1.2.*

□

Zovšeobecnené Petriho siete sa tiež označujú, ako *PT siete* (Place - Transition nets), pričom podľa niektorých definícií (napríklad v [26]) miesta v nich môžu, ale nemusia, mať definovanú kapacitu.

## 2.5 ER, TER a TB siete

### 2.5.1 ER A TER siete

*ER* a *TER siete* patria medzi tie modifikácie Petriho sietí, ktorých vyjadrovacia sila je väčšia ako u obyčajných Petriho sietí. Značky (tokeny) v týchto sieťach majú individualitu, ktorá je daná vektorom hodnôt premenných z množiny $ID$, ktoré sú definované nad určitou množinou hodnôt $V$. Vykonanie prechodu je podmienené reláciou $act_t$. Po realizácii prechodu majú generované značky nové hodnoty z množiny $V$. TER siete umožňujú pri modelovaní systémov zaradiť pojem času.

**ER siete** (Environment-Relationship Nets) možno charakterizovať nasledovne [12]:

- Značky (tokeny) sú prostredia na $ID$ a $V$. Univerzálna množina prostredí $ENV = V^{ID}$. $ID$ je množina identifikátorov a $V$ množina hodnôt, ktoré môžu identifikátory nadobúdať.
- Každý prechod je zviazaný s akciou. Akcia je relácia:

$$\alpha(t) \subseteq ENV^{k(t)} \times ENV^{h(t)} \text{ a } k(t) = |{}^\bullet t|, h(t) = |t^\bullet|$$

  $\pi(t)$ značí projekciu $\alpha(t)$ na $ENV^{k(t)}$ a nazýva sa *predikátom* $t$.
- Značenie je priradenie multimnožín[4] prostredí k miestam.
- Prechod $t$ je aktivovaný pri značení $m$ práve vtedy, ak $\forall p_i \in^\bullet t$ existuje minimálne jedna značka $env_i$ taká, že $< env_1, ..., env_{k(t)} > \in \pi(t)$.
- Realizácia prechodu je trojica $x =< enab, t, prod > : < enab, prod > \in \alpha(t)$.
- Výskyt realizácie prechodu v značení $m$ spôsobí zmenu značenia z $m$ na $m'$, odstránením $enab$ z príslušných vstupných miest $t$.
- Pre ER siete je ďalej možné definovať sekvenciu aktivácií a sekvenciu prípustných značení. Zápis $m \overset{x}{\vdash} m'$ značí, že $x =< enab, t, prod >$ je aktivácia, ktorá produkuje značenie $m'$ z $m$.

**TER siete** (Time ER Nets) sú časovým rozšírením ER sietí. Z ER sietí vznikli pridaním špeciálnej premennej *chronos* do prostredia. *Chronos* reprezentuje čas vzniku prostredia. Notácia $env_i.chronos$ je použitá na prístup k premennej *chronos* prostredia $env_i$. Realizácia prechodu je štruktúra $x =< enab, t, prod >, enab =< env_1, \ldots, env_{k(t)} >,\ prod =< env'_1, \ldots, env'_{h(t)} >$.

Každá akcia musí spĺňať tieto axiómy:

**Axióma 1:** Pre ľubovoľnú realizáciu prechodu $env'_j.chronos \geq env_i.chronos$, pre každé $i, j$.

**Axióma 2:** Pre ľubovoľnú realizáciu $x$, existuje hodnota označená $time(x)$ (čas realizácie) taká, že $env'_j.chronos = time(x), \forall j(0 \leq j \leq h(t))$.

**Axióma 3:** Pre ľubovoľnú sekvenciu realizácií $s = x_1 x_2 ... x_{|s|}$, $time(x_i) \leq time(x_j)$ práve vtedy ak $i < j, 1 \leq i, j \leq |s|$.

[4]Multimnožina nad neprázdnou množinou $E$ je funkcia $x : E \to \mathbb{N}$ Hodnota $x(e)$ je počet výskytov (koeficient) prvku $e \in E$ v multimnožine $x$.

ER sieť, ktorá pre každé prostredie obsahuje premennú *chronos* a platia axiómy 1 a 2 sa nazýva TER sieť.

Pre danú ER sieť, ktorá spĺňa axiómu 2, sekvenciu realizácií $s = x_1x_2...x_s$ nazývame *časovo usporiadanou*, ak $\forall i, j, i < j \Rightarrow time(x_i) \leq time(x_j)$.

Pre daný prechod $t$ a *enab* pre tento prechod TER siete je možné definovať množinu prípustných časov realizácie *f-time* (set of possible firing times):

$$f\text{-}time(t, enab) = \{t| < enab, prod > \in \alpha(t), t = prod.chronos\}$$

Ak $s = x_1x_2...x_s$ je sekvencia realizácií TER siete s počiatočným značením $m_0$ a $m_0 \overset{s}{\vdash} m$, potom sekvencia $s$ je *silnou* (strong), práve vtedy ak je časovo usporiadaná a $\forall t' \in T$ neexistuje $enab_i'$ pre $t'$ v $m_i$ také, že $time(x_{i+1}) > \sup(f\text{-}time(t, enab_i'))$.
Pre TER siete môže platiť axióma:

**Axióma 3':** Všetky sekvencie realizácií sú silné.

TER sieť, ktorá spĺňa axiómy 1,2 a 3' je *silná* TER sieť.

### 2.5.2 TB siete

TB siete (Time-Basic Nets) sú zjednodušením TER sietí. Jedinou premennou prostredí je *chronos*, preto prostrediam v TB sieťach tiež hovoríme *časové značky*. Každý prechod má definovanú časovú funkciu, ktorá definuje vzťah medzi *chronos*-mi značiek konzumovaných prechodom a vznikajúcich značiek.

#### Definícia TB siete

TB sieť je vektor $(P, T, F, \Theta, tf, m_0)$, kde:

- $P, T, m_0$ sú definované ako v bežných PS. Každá značka z $m_0$ má však navyše definovanú premennú *chronos*.
- $F$ je konečná množina orientovaných hrán siete, pričom preset prechodu $t$ (označujeme ${}^\bullet t$) tvorí množina miest spojených orientovanou hranou vstupujúcou do prechodu $t$ a postset prechodu $t$ (označujeme $t^\bullet$) tvorí množina miest spojených orientovanou hranou vychádzajúcou z t.[5]
- $\Theta$ predstavuje množinu prípustných hodnôt pre premennú *chronos*.

[5]Význam množiny $F$ je totožný s významom funkcií *pre* a *post* v bežných Petriho sieťach.

- $tf$ je množina časových funkcií spojených s prechodmi TB siete ($tf = \{tf_t | t \in T\}$). Nech $en$ označuje aktivačnú n-ticu[6] (enabling tuple), tzn. skupinu tokenov, jeden pre každé miesto v ${}^\bullet t$. Funkcia $tf_t$ priraďuje každej n-tici $en$ množinu hodnôt $\theta, \theta \subseteq \Theta$.[7] Časové značky tokenov vzniknutých odpálením prechodu $t$ majú hodnotu zhodnú s časom odpálenia prechodu $t$.

TB siete možno rozdeliť do dvoch tried:

- WTS TB siete (Weak Time Semantics) - *slabé* časové sémantiky, kde TB siete spĺňajú axiómy 1,2 a 3 z kapitoly 2.5.1. V praxi to znamená, že prechod nemusí byť realizovaný v rámci daného časového intervalu (je možný výskyt inej udalosti). Vlastnosť je vhodná na opis udalostí, ktoré môžu, ale nemusia nastať v určitom čase.

- STS TB siete (Strong Time Semantics) - *silné* časové sémantiky, kde patria TB siete, ktoré spĺňajú axiómy 1,2 a 3' z kapitoly 2.5.1.

[6] V TER sieťach sme aktivačnú n-ticu označovali *enab*.

[7] Časová funkcia $tf_t$ definuje vzťah medzi chronosmi značiek konzumovaných prechodom $t$ a chronosmi značiek vznikajúcich pri realizácii $t$.

## 2.6 Farbené Petriho siete

*Farbené Petriho siete* (Coloured Petri Nets, CPN) predstavujú modifikáciu GPN, vyznačujúcu sa značnou modelovacou silou a analytickými vlastnosťami. Ich prvá verzia ($CP^{81}$ siete) vznikla v roku 1981 ako odpoveď na isté problémy pri analýze predikátovo-prechodových (PrT) sietí. $CP^{81}$ siete síce tento problém riešili, no výrazy na hranách v týchto sieťach boli horšie čitateľné a pochopiteľné ako tie v PrT sieťach. Keďže si boli PrT a $CP^{81}$ siete veľmi podobné, vznikol kombinovaný model, nazvaný vysokoúrovňové Petriho siete (HL-nets). Tento názov sa však začal používať aj pre PrT a $CP^{81}$ siete, preto boli HL-nets premenované na farbené Petriho siete (CPN). Pre CPN bol vytvorený modelovací, simulačný a analylický nástroj *CPN Tools*, ktorý je pokračovaním nástroja *Design/CPN* a je voľne dostupný z [35].

Základným rozdielom nedzi CPN a GPN je, že v CPN sú značky (tokeny) individualizované - každý má svoju typ (*množinu farieb*) a hodnotu (*farbu*). Počet, typ a podmienky kladené na tokeny, ktoré sa zúčastňujú realizácie prechodov sú vyjadrené pomocou *sieťových výrazov*, kde patria *hranové výrazy* (priradené hranám siete) a *stráže* (priradené prechodom siete). Neindividualizované tokeny majú v CPN priradený typ *unit* obsahujúci jedinú hodnotu ().

V tejto časti je stručne uvedená definícia nehierarchickej CPN a jej správania. Definície sú prevzaté z [16], kde možno nájsť aj ich podrobnejšie vysvetlenie, vrátane príkladu, a z [18], čo je druhá časť trojzväzkového diela [17], [18], [19], ktoré sa farbenými Petriho sieťami zaoberá podrobne. Niektoré výrazy, ktoré v definíciách budú použité sú v tabuľke 2.1.

| **Výraz** | **Význam** |
|---|---|
| $\mathbb{N}$ | Množina prirodzených čísel vrátane nuly. |
| $\mathbb{B}$ | Booleovský typ, $\mathbb{B} = \{true, false\}$. |
| $[S \to T]$ | Množina všetkých funkcií z $S$ do $T$. $S$ a $T$ sú množiny. |
| $[S \to T]_L$ | Množina všetkých lineárnych funkcií z $S$ do $T$. |
| $Type(e)$ | Typ výrazu $e$. |
| $Var(e)$ | Množina premenných vo výraze $e$. |

Tabuľka 2.1: Výrazy používané v definíciách k CPN

Pred uvedením definície CPN je nutné uviesť definíciu *multi-množín*, ktoré sú používané na vyjadrenie značení a taktiež v sieťových výrazoch.

**Definícia 2.6.1.** Multi-množina *$m$ nad neprázdnou množinou $S$, je funkcia*

*$m \in [S \to \mathbb{N}]$ ktorú reprezentujeme ako formálny súčet:*

$$\sum_{s \in S} m(s)'s$$

*Ako $S_{MS}$ označujeme množinu všetkých multi-množín nad S. Čísla z $\{m(s) | \ s \in S\}$, sú* koeficienty *multi-množiny. Platí, že $s \in m \Leftrightarrow m(s) \neq 0$.*

Číslo $m(s)$ teda udáva počet výskytov prvku $s$ v multi-množine $m$. Napríklad multi-množina $2`a + +3`j + +1`k$ je vlastne $\{a, a, j, j, j, k\}$. V angličtine sa pre multimnožinu používa aj výraz "bag". Pre multi-množiny existuje niekoľko štandardných operátorov:

**Definícia 2.6.2.** Sčítanie *(2.1),* skalárne násobenie *(2.2),* porovnanie *(2.3, 2.4),* veľkosť *(2.5) a* rozdiel *(2.6) sú definované pre všetky $m, m1, m2 \in S_{MS}$ a všetky $n \in \mathbb{N}$ takto:*

$$m_1 + m_2 = \sum_{s \in S} (m_1(s) + m_2(s))'s \tag{2.1}$$

$$n * m = \sum_{s \in S} (m * m(s))'s \tag{2.2}$$

$$m_1 \neq m_2 \equiv \exists s \, (s \in S) : m_1(s) \neq m_2(s) \tag{2.3}$$

$$m_1 \leq m_2 \equiv \forall s \, (s \in S) : m_1(s) \leq m_2(s) \tag{2.4}$$

$$|m| = \sum_{s \in S} m(s) \tag{2.5}$$

$$m_1 - m_2 = \sum_{s \in S} (m_1(s) - m_2(s))'s \tag{2.6}$$

*Rozdiel (2.6) je definovaný len ak $m_2 \leq m_1$.*

Operátory porovnania $=$, $\geq$, $>$, $<$ sú definované rovnakým spôsobom, ako $\neq$ a $\leq$ v (2.3, 2.4).

### 2.6.1 Definícia farbenej Petriho siete

**Definícia 2.6.3.** *Farbená Petriho sieť je 9-tica*

*$PN_C = (\Sigma, P, T, A, N, C, G, E, I)$, kde:*

- $\Sigma$ *je konečná množina neprázdnych* typov *(*množín farieb*),*
- $P$ *je konečná množina* miest,
- $T$ *je konečná množina* prechodov,

- *A je konečná množina* hrán*, takých že:*
  $P \cap T = P \cap A = T \cap A = \varnothing$,
- *N je* funkcia uzlov*. Je definovaná z A do* $P \times T \cup T \times P$,
- *C je* funkcia farieb*. Je definovaná z P do* $\Sigma$,
- *G je* funkcia stráží*. Je definovaná z T do výrazov takých, že:*
  $\forall t\,(t \in T) : (Type(G(t)) = \mathbb{B} \wedge Type(Var(G(t))) \subseteq \Sigma)$,
- *E je* funkcia hranových výrazov*. Je definovaná z A do výrazov takých, že:* $\forall a\,(a \in A) : (Type(E(a)) = C(p)_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma)$, *kde* $N(a) = (p,t) \vee N(a) = (t,p)$,
- *I je* inicializačná funkcia*. Je definovaná z P do uzavretých výrazov takých, že:* $\forall p\,(p \in P) : Type(I(p)) = C(p)_{MS}$.

Množina $\mathbb{B}$ a funkcie $Type$ a $Var$ sú definované v tabuľke 2.1. Uzavretý výraz je taký, v ktorom sa nevyskytujú premenné.

Aby bola definícia CPN úplná, je nutné zadefinovať jazyk, v ktorom sa špecifikujú sieťové výrazy. Pre CPN, ako sú používané v CPN Tools sa používa funkcionálny programovací jazyk *SML*, presnejšie jeho mierne modifikovaná varianta nazvaná *CPN ML*. Môže však ísť aj o iný jazyk, no musí spĺňať požiadavky uvedené v [16].

V nasledujúcich definíciách bude použitá aj takáto notácia, pre všetky $t \in T$ a $(x_1, x_2) \in P \times T \cup T \times P$:

- $A(t) = \{a \in A|\ N(a) \in (P \times \{t\}) \cup (\{t\} \times P)\}$,
- $Var(t) = \{v|\ v \in Var(G(t)) \vee \exists a.(a \in A(t) \wedge v \in Var(E(a)))\}$,
- $A(x_1, x_2) = \{a \in A|\ N(a) = (x_1, x_2)\}$,
- $E(x_1, x_2) = \sum_{a \in A(x_1,x_2)} E(a)$,
- $Expr{<}b{>}$ je hodnota získaná vyhodnotením výrazu $Expr$ pri priradení hodnôt premenným určeným väzbou $b$.

### 2.6.2 Výpočet farbenej Petriho siete

**Definícia 2.6.4.** Väzba *(binding) prechodu t je funkcia b definovaná na* $Var(t)$*, taká že*

- $\forall v\,(v \in Var(t)) : b(v) \in Type(v)$ *a*
- $G(t){<}b{>} = true$.

$B(t)$ *je množina všetkých väzieb prechodu t.*

**Definícia 2.6.5.** Tokenový element *(token element) je dvojica* $(p, c)$*, kde* $p \in P$ *a* $c \in C(p)$. Väzobný element *(binding element) je dvojica* $(t, b)$*, kde* $t \in T$ *a* $b \in B(t)$.

$TE$ *je množina všetkých tokenových elementov a* $BE$ *je množina všetkých väzobných elementov.*

**Definícia 2.6.6.** Značenie *(marking) je multi-množina nad* $TE$ *a* krok *je neprázdna konečná multi-množina nad* $BE$. $\mathbb{M}$ *je množina všetkých značení a* $\mathbb{Y}$ *je množina všetkých krokov.*

Počiatočné značenie $M_0$ *je značenie, ktoré vznikne ohodnotením inicializačných výrazov:* $\forall(p, c)\,((p, c) \in TE) : M_0(p, c) = (I(p))(c)$.

Každé značenie $M \in TE_{MS}$ určuje jedinečnú funkciu $M^*$ definovanú na $P$, takú že $M^*(p) \in C(p)_{MS}$:

$$\forall p \forall c\,(p \in P \wedge c \in C(p)) : (M^*(p))(c) = M(p, c)$$

Na druhej strane každá funkcia $M^*$, definovaná na $P$, taká že $M^*(p) \in C(p)_{MS}$ pre všetky $p \in P$, určuje jedinečné značenie $M$:

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

Preto budeme často reprezentovať značenia, ako funkcie definované na $P$ a budeme používať rovnaké meno $(M)$ pre funkčnú $(M^*)$ aj multi-množinovú $(M)$ reprezentáciu značenia.

**Definícia 2.6.7.** *Krok* $Y$ *je* realizovateľný *(enabled) v značení* $M$ *ak platí:*

$$\forall p\,(p \in P) : \sum_{(t,b) \in Y} E(p, t) < b > \quad \leq M(p)$$

*Potom hovoríme, že* $(t, b)$ *je realizovateľný a tiež, že* $t$ *je realizovateľný. Elementy kroku* $Y$ *sú* súbežne realizovateľné *(concurrently enabled), ak* $|Y| \geq 1$.

*Ak* $Y$ *je realizovateľný v značení* $M_1$*, môže sa realizovať, čím sa značenie zmení z* $M_1$ *na* $M_2$*, definované takto:*

$$\forall p\,(p \in P) : M_2(p) = \Big(M_1(p) - \sum_{(t,b) \in Y} E(p, t) < b > \Big) + \sum_{(t,b) \in Y} E(t, p) < b >$$

*Hovoríme, že* $M_2$ *je* priamo dosiahnuteľný *(directly reachable) z* $M_1$*, čo zapisujeme ako* $M_1[Y > M_2$ *alebo ako* $M_1 \overset{Y}{\vdash} M_2$.

Namiesto výrazu "realizácia" (kroku alebo prechodu) tiež používame výrazy "výskyt" alebo "odpálenie".

**Definícia 2.6.8.** Konečná postupnosť výskytov *(finite occurrence sequence) je postupnosť značení a krokov:*

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

*taká, že* $n \in \mathbb{N}$*, a* $M_i[Y_i > M_{i+1}$ *pre každé* $i \in \{1, 2, \ldots, n\}$*.* $M_1$ *je štartovacie značenie,* $M_{n+1}$ *koncové a* $n$ *je dĺžka postupnosti.*

*Značenie* $M''$ *je* dosiahnuteľné *zo značenia* $M'$ *práve vtedy ak existuje konečná postupnosť výskytov začínajúca v* $M'$ *a končiaca v* $M''$*.*

*Množinu značení dosiahnuteľných z* $M$ *označujeme* $[M>$*. Značenie je dosiahnuteľné, práve vtedy ak patrí do* $[M_0>$*.*

### 2.6.3 Analytické vlastnosti farbenej Petriho siete

Farbené Petriho siete majú rovnakú vyjadrovaciu silu, ako zovšeobecnené Petriho siete a teda aj ako obyčajné Petriho siete. Každú CPN, ktorej $\Sigma$ obsahuje len konečné množiny, môžeme previesť na výpočtovo ekvivalentnú GPN. Ak by bola množina niektorá množina farieb nekonečná (tzn. $\Sigma$ by obsahovala aj nekonečnú množinu), potom by sme pri pokuse o prevod dospeli ku GPN s nekonečnou štruktúrou.

Keďže majú CPN aj GPN rovnakú vyjadrovaciu silu, mali by byť aj rovnako analyzovateľné. Naozaj, aj pre CPN existujú metódy pre automatický výpočet invariantov miest (S-invarianty) a prechodov (T-invarianty), ktoré sú stručne popísané v tejto časti. Tie vychádzajú z metód pre GPN siete, no ich výpočet je oveľa komplikovanejší.

Najprv uvedieme reprezentáciu CPN incidenčnou maticou.

**Definícia 2.6.9.** Incidenčná matica *CPN je matica* IM *typu* $m \times n$*, kde* $m = |P|$ *a* $n = |T|$*, s prvkami, ktoré sú funkciami:*

$$\begin{aligned} IM(p,t) &\in [B(t)_{WS} \to C(p)_{WS}]_L \\ IM(p,t)(b) &= E(t,p)<b> - E(p,t)<b>,\ b \in B(t) \end{aligned}$$

Výraz $E(t,p) < b > -E(p,t) < b >$ je tzv. *váženou množinou* (weighted set), ktorá je definovaná ako multi-množina, kde sú povolené aj záporné koeficienty. Operácia odčítania je tu vždy realizovateľná a môžeme násobiť aj záporným číslom. Množinu všetkých vážených množín nad množinou $A$ označíme $A_{WS}$.

**Definícia 2.6.10.** S-invariant *nehierarchickej CPN je množina funkcií*

$$Ws = \{W_p | p \in P\},\ kde\ W_p \in [C(p) \to A_{WS}]_L,\ A \in \Sigma$$

*taká, že*

$$\forall M\,(M \in [M_0>) : \sum_{p \in P} W_p(M(p)) = \sum_{p \in P} W_p(M_0(p))$$

Funkcie $W_p$ sú tzv. *váhy miest* (place weights). Sú definované ako $W_p \in [C(p) \to A_{WS}]_L$, kde $A \in \Sigma$ je typ zdieľaný všetkými váhami. Funkciu $W_p$ možno rozšíriť na $\underline{W_p} \in [C(p)_{WS} \to A_{WS}]_L$, definovanú ako:

$$\underline{W_p}(m) = \sum_{c \in C(p)} m(c) * W_p(c),\ m \in C(p)_{WS}$$

Keďže nie je nutné rozlišovať medzi $W_p$ a jej lineárnym rozšírením $\underline{W_p}$ , budeme obe označovať $W_p$.

Samotný S-invariant získame ako riešenie rovnice

$$WS * IM = \vec{0}$$

kde

- $\vec{0}$ je stĺpcový vektor nulových funkcií, ktoré každý argument zobrazia do $\varnothing$ ($\varnothing$ je prázdna vážená množina),
- $IM$ je icidenčná matica,
- $*$ je kompozícia funkcií $W_p$ a $IM(p,t)$,
- $WS$ je riadkový vektor, ktorého prvkami sú funkcie z množiny $Ws$ (je to vlastne vektorová podoba $Ws$).

Vektor $WS$ je riešením rovnice a určuje S-invariant.

T-invariant je definovaný analogicky k S-invariantu, aj keď v tomto prípade nie je invariantom priamo množina váhových funkcií.

**Definícia 2.6.11.** *Pre nehierarchickú CPN* množina váh prechodov *(transition weights) s doménou $A \in \Sigma$ je množina funkcií*

$$Wtr = \{W_t | t \in T\},\ kde\ W_t \in [A_{WS} \to B(t)_{WS}]_L$$

.

*$Wtr$ je* prechodový tok *(transition flow) práve vtedy ak:*

$$\forall a \forall p\,(a \in A \wedge p \in P) : \sum_{(t,b) \in Wtr(a)} E(p,t)<b> = \sum_{(t,b) \in Wtr(a)} E(t,p)<b>$$

*Konečná multi-množina $Y \in BE_{MS}$ je* T-invariant *práve vtedy ak:*

$$\forall p\,(p \in P) : \sum_{(t,b) \in Y} E(p,t)<b> = \sum_{(t,b) \in Y} E(t,p)<b>$$

$Wtr(a)$ je množina $Wtr(a) = \{Wtr(a)|\ t \in T \wedge a \in A_{WS}\}$. Vzťah prechodového toku a T-invariantu vyjadruje nasledujúca veta:

**Veta 2.6.1.** *$Wtr$ je prechodový tok práve vtedy, ak pre každé $a \in A$ je $Wtr(a)$ T-invariant.*

Prechodové toky, z ktorých určíme T-invarianty, dostaneme ako riešenie rovnice

$$IM * WT = \vec{0}$$

kde

- $\vec{0}$ je stĺpcový vektor nulových funkcií, ktoré každý argument zobrazia do $\varnothing$,
- $IM$ je icidenčná matica,
- $*$ je kompozícia funkcií $IM(p,t)$ a $W_t$,
- $WT$ je riadkový vektor, ktorého prvkami sú funkcie z množiny $Wtr$ (je to vlastne vektorová podoba $Wtr$).

Riešením rovnice je vektor $WT$, z ktorého určíme T-invarianty.

## 2.7 Hodnotiace Petriho siete

*Hodnotiace Petriho siete* (Evaluative Petri Nets, EPN) sú rozšírením obyčajných Petriho sietí, ktoré bolo predstavené v [10]. Vyznačujú sa vyjadrovacou silou rovnou Turingovým strojom a cennými analytickými vlastnosťami. Tu uvádzame iba základné definície[8] týkajúce sa EPN a ich správania, viac informácií je možné nájsť v [10]. V nasledujúcom texte budeme označovať množinu prirodzených čísel vrátane nuly ako $\mathbb{N}$ a množinu celých čísel ako $\mathbb{Z}$.

**Definícia 2.7.1.** Hodnotiaca Petriho sieť *(EPN) je 5-tica*

$H = (P_H, T_H, pre_H, post_H, m_0)$, *kde*

$P_H = P \cup P_F \cup P_E$ *je konečná množina miest:*
$P = \{p_1, \ldots, p_k\}$ *sú individuálne premenné,* $k = |P|$,
$P_F = \{fn_1, \ldots, fn_v\},\ v = |P_F|$,
$\forall ii(1 \leq ii \leq v) : fn_{ii} = f_{ii}(p_1, \ldots, p_k), f_{ii} \in F$,
$P_E = \{pr_1, \ldots, pr_w\},\ w = |P_E|$,

[8] Definície uvedené v tejto časti sú oproti práci [10] mierne modifikované.

$\forall jj(1 \leq jj \leq w) : pr_{jj} = e_{jj}(p_1, \ldots, p_k), e_{jj} \in E;$

$T_H = \{t_1, \ldots, t_{|T_H|}\}$ *je konečná množina prechodov;*

$pre : P_H \times \Omega \times T_H \rightarrow \mathbb{N} \times \mathbb{C}$
$post : P_H \times \Omega \times T_H \rightarrow \mathbb{N} \times \mathbb{C}$
*sú funkcie definujúce prepojenie miest a prechodov;*

$m_0$ *je počiatočné značenie.*

*F je množina funkcií s oborom hodnôt* $\mathbb{N} - \{0\}$*, E je množina predikátov,* $\Omega = P \cup P_F \cup \{1\}$*,* $\mathbb{C} = \mathbb{K} \cup \{\omega, d\}$*,* $\mathbb{K} \subseteq \mathbb{N}$*. Symboly* $\omega$ *a d nepatria do* $P_H \cup T_H$*.*

□

Symbol $\omega$ predstavuje číslo väčšie, ako ktorékoľvek prirodzené číslo. Symbol $d$ má nasledujúci význam: ak $pre(X, Y, t) = (n, d)$ alebo $post(X, Y, t) = (n, d)$, potom $m(X)$ môže nadobúdať záporné hodnoty.

Všetky Hodnotiace Petriho siete tvoria triedu $EPN_{cl}$. Všetky miesta Hodnotiacich Petriho sietí tvoria triedu $P_{H_{cl}}$, ktorá obsahuje 3 (pod)triedy - $P_{cl}$, $P_{F_{cl}}$ and $P_{E_{cl}}$, a všetky prechody tvoria triedu $T_{H_{cl}}$.

Trieda hodnotiacich Petriho sietí v sebe zahŕňa triedu obyčajných PS aj ich základných modifikácií, uvedených v časti 2.4. Obyčajné PS a ich základné modifikácie sú také EPN, ktoré majú $P_H = P$, $Y = \{1\}$ a kde platí

$$\forall p \forall t \forall \nu \, (p \in P \wedge t \in T_H \wedge \nu \in \mathbb{N}) : pre(p, 1, t) \neq (\nu, d) \wedge post(p, 1, t) \neq (\nu, d)$$

Podľa toho, aké sú hodnotu funkcií *pre* a *post* potom môže ísť o kapacitne obmedzenú, zovšeobecnenú, alebo obyčajnú PS.

Stav EPN je vyjadrený jej značením:

**Definícia 2.7.2.** Značenie *EPN H je funkcia* $m : P_H \rightarrow \mathbb{Z}$*.*

- *Ak* $pl \in P$*, potom* $m(pl)$ *je počet tokenov v mieste pl.*
- *Ak* $pl \in P_F$*,* $pl = fn = f(p_1, \ldots, p_k)$*, potom* $m(pl) = f(m(p_1), \ldots, m(p_k))$*.*
- *Ak* $pl \in P_E$*,* $pl = pr = e(p_1, \ldots, p_k)$*, potom* $m(pl) = 1 \Leftrightarrow e(m(p_1), \ldots, m(p_k)) = true$*,* $m(pl) = 0 \Leftrightarrow e(m(p_1), \ldots, m(p_k)) = false$*.*

Vektorová forma $m$ *je vektor* $\vec{m} \in \mathbb{Z}^{|P_H|}$,

$$\begin{aligned} \vec{m} &= (m(pl_1), m(pl_2), \ldots, m(pl_{|P_H|})) \\ &= (m(p_1), \ldots, m(p_k), m(fn_1), \ldots, m(fn_v), m(pr_1), \ldots, m(pr_w)) \end{aligned}$$

□

Vektorovú formu značenia budeme v ďalšom zapisovať bez šípky, tzn. ako $m$.

Značenie $m$ je možné rozdeliť na dve časti. Prvá bude pozostávať zo značení miest z $P$ a druhá zo značení miest z $P_F \cup P_E$. Prvá časť, ktorú nazveme *nezávislé značenie*, jednoznačne určuje celé značenie EPN: druhá časť (*závislé značenie*) je iba funkciou prvej, definovanou miestami-funkciami z $P_F$ a miestami-predikátmi z $P_E$.

**Definícia 2.7.3.** *Nech $H$ je EPN a $m$ je jej značenie. Potom* nezávislé značenie $H$ *je vektor* $\iota(m) = (m(p_1), \ldots, m(p_k))$, $p_i \in P$.
Nezávislé počiatočné značenie $H$ *je vektor* $\iota(m_0) = (m_0(p_1), \ldots, m_0(p_k))$, $p_i \in P$.

□

### 2.7.1 Výpočet EPN

V definíciách týkajúcich sa výpočtu EPN budú používané množiny ${}^\bullet t$, $t^\bullet$, funkcie $Pr_\Omega(X,t)$, $Pr^{\bar{d}}_\Omega(X,t)$, $Ps_\Omega(X,t)$, $Ps^k_\Omega(X,t)$, kde $X \in P_H$, $t \in T_H$, a funkcia $M(Y)$, kde $Y \in \Omega$. Tieto sú definované nasledovne:

$$\begin{aligned} {}^\bullet t &= \{X \in P_H |\ \exists Y \in \Omega\ :\ pre(X,Y,t) \neq (0,0)\} \\ t^\bullet &= \{X' \in P_H |\ \exists Y' \in \Omega\ :\ post(X',Y',t) \neq (0,0)\} \\ Pr_\Omega(X,t) &= \{Y | Y \in \Omega \wedge pre(X,Y,t) = (\nu_Y, c_Y) \wedge \nu_Y > 0\} \\ Pr^{\bar{d}}_\Omega(X,t) &= \{Y | Y \in \Omega \wedge pre(X,Y,t) = (\nu_Y, c_Y) \wedge \nu_Y > 0 \wedge\ c_Y \neq d\} \\ Ps_\Omega(X,t) &= \{Y' | Y' \in \Omega \wedge post(X,Y',t) = (\nu_{Y'}, c_{Y'}) \wedge \nu_{Y'} > 0\} \\ Ps^k_\Omega(X,t) &= \{Y' | Y' \in \Omega \wedge post(X,Y',t) = (\nu_{Y'}, c_{Y'}) \wedge \nu_{Y'} > 0 \wedge c_{Y'} \in \mathbb{K}\} \\ M(Y) &= \begin{cases} 1 & if\ Y = 1 \\ m(Y) & if\ Y \in P \cup P_F \end{cases} \end{aligned}$$

**Definícia 2.7.4.** *Prechod $t \in T_H$* je vykonateľný v značení $m$ *(čo označujeme $m \overset{t}{\vdash}$) hodnotiacej Petriho siete $H$ práve vtedy, ak platia nasledujúce podmienky:*

$$\begin{gathered} \forall X \in {}^\bullet t : Pr^{\bar{d}}_\Omega(X,t) \neq \varnothing \Rightarrow M(X) \geq \textstyle\sum_{\forall Y \in Pr^{\bar{d}}_\Omega(X,t)} \nu_Y . M(Y) \\ pre(X,Y,t) = (\nu_Y, c_Y) \end{gathered} \tag{2.7}$$

$$\forall X', Y'(X' \in t^{\bullet} \wedge Y' \in Ps^{k}_{\Omega}(X', t) \wedge post(X', Y', t) = (\nu_{Y'}, c_{Y'})) : \\ M(X') + \nu_{Y'}.M(Y') \leq c_{Y'} \tag{2.8}$$

*Ak $t$ je vykonateľný v $m$, môže byť vykonaný (odpálený). Vykonanie $t$ v $m$ vedie k novému značeniu $m'$ (čo označujeme $m \overset{t}{\vdash} m'$):*

$$\forall X \in P_H : m'(X) = m(X) - \sum_{\forall Y \in Pr_{\Omega}(X,t)} \nu_Y.m(Y) + \sum_{\forall Y' \in Ps_{\Omega}(X,t)} \nu_{Y'}.m(Y') \\ pre(X, Y, t) = (\nu_Y, c_Y),\ post(X, Y', t) = (\nu_{Y'}, c_{Y'}) \tag{2.9}$$

□

**Definícia 2.7.5.** *Sekvencia $\sigma \in T^*_H$, $\sigma = t_{i_1}, t_{i_2}...t_{i_r}$ je* prípustná sekvencia prechodov *(psp)* EPN $H$ v $m_0$*, práve vtedy, ak existuje sekvencia značení $m_0, m_1...m_r$ taká, že $\forall j(1 \leq j \leq r) : m_{j-1} \overset{t_{i_j}}{\vdash} m_j$.*

□

### 2.7.2 Príklad

Tu uvedená *EPN Divider* je prevzatá z [10] a simuluje celočíselné delenie dvoch čísel, $x1$ a $x2$. Po postupnom odpálení všetkých vykonateľných prechodov sa *EPN Divider* dostane do mŕtveho značenia $m_f$, kde:

$$\begin{aligned} m_f(p_8) &= m_0(p_6)\ div\ m_0(p_7) = x1\ div\ x2 \\ m_f(p_9) &= m_0(p_6)\ mod\ m_0(p_7) = x1\ mod\ x2 \end{aligned}$$

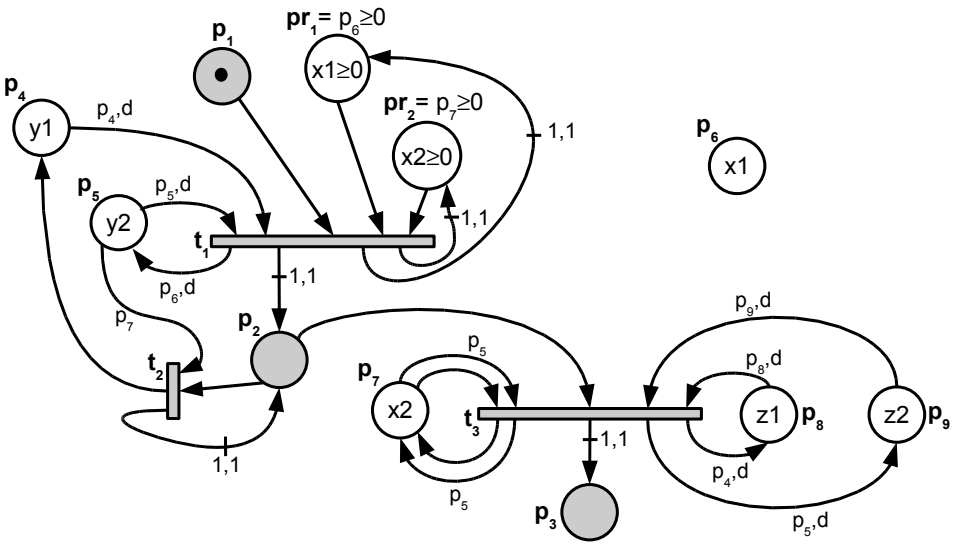
Definícia *EPN Divider* je v tabuľke 2.2 a jej graf je na obrázku 2.4.

$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$, $P_F = \varnothing$, $P_E = \{pr_1, pr_2\}$
$pr_1 = p_6 \geq 0$, $pr_2 = p_7 \geq 0$
$T_H = \{t_1, t_2, t_3\}$
$m_0 = (1, 0, 0, y1, y2, x1, x2, z1, z2, x1 \geq 0, x2 \geq 0)$

| p | o | t | pre(p,o,t) |
|---|---|---|---|
| $p_1$ | 1 | $t_1$ | $(1, \omega)$ |
| $p_2$ | 1 | $t_2$ | $(1, \omega)$ |
| $p_2$ | 1 | $t_3$ | $(1, \omega)$ |
| $p_4$ | $p_4$ | $t_1$ | $(1, d)$ |
| $p_5$ | $p_5$ | $t_1$ | $(1, d)$ |
| $p_5$ | $p_7$ | $t_2$ | $(1, \omega)$ |
| $p_7$ | $p_5$ | $t_3$ | $(1, \omega)$ |
| $p_7$ | 1 | $t_3$ | $(1, \omega)$ |
| $p_8$ | $p_8$ | $t_3$ | $(1, d)$ |
| $p_9$ | $p_9$ | $t_3$ | $(1, d)$ |
| $pr_1$ | 1 | $t_1$ | $(1, \omega)$ |
| $pr_2$ | 1 | $t_1$ | $(1, \omega)$ |
| inak | | | $(0, 0)$ |

| p | o | t | post(p,o,t) |
|---|---|---|---|
| $p_2$ | 1 | $t_1$ | $(1, 1)$ |
| $p_3$ | 1 | $t_3$ | $(1, 1)$ |
| $p_4$ | 1 | $t_2$ | $(1, \omega)$ |
| $p_5$ | $p_6$ | $t_1$ | $(1, \omega)$ |
| $p_7$ | $p_5$ | $t_3$ | $(1, \omega)$ |
| $p_7$ | 1 | $t_3$ | $(1, \omega)$ |
| $p_8$ | $p_4$ | $t_3$ | $(1, d)$ |
| $p_9$ | $p_5$ | $t_3$ | $(1, d)$ |
| $pr_1$ | 1 | $t_1$ | $(1, 1)$ |
| $pr_2$ | 1 | $t_1$ | $(1, 1)$ |
| $p_2$ | 1 | $t_2$ | $(1, 1)$ |
| inak | | | $(0, 0)$ |

Tabuľka 2.2: Definícia *EPN* Divider.




Obr. 2.4: Graf EPN Divider

## 2.8 Problém dosiahnuteľnosti a jeho riešenie

### 2.8.1 Definícia problému dosiahnuteľnosti

*Problém dosiahnuteľnosti* (Reachability problem - *RP*) v Petriho sieťach je definovaný nasledovne [12]:

**Definícia 2.8.1.** *Majme PS* $N_0 = (P, T, pre, post, m_0)$ *a* $k$*-rozmerný nezáporný celočíselný vektor* $q$ *(*$k = |P|$*). Problém či* $q \in R(N_0)$ *je nazývaný* problémom dosiahnuteľnosti *(*inštanciou problému dosiahnuteľnosti*) Petriho siete* $N_0$ *(pre stav* $q$*).*

Tento problém značnej časovej zložitosti, ktorý zaraďujeme medzi takzvané nezvládnuteľné (intractable) problémy, má kľúčové postavenie medzi algoritmickými problémami Petriho sietí, pretože riešenie problémov ako sú živosť, ohraničenosť, pokrytie, deadlock, úzko súvisí práve s RP [12].
*Problém časovej dosiahnuteľnosti* (Time reachability problem - *TRP*) je rozšírením RP pre časovo-kritické systémy. To znamená, že nás nezaujíma len to, či nejaký (domnelý) stav je dosiahnuteľný, ale aj to, v akom čase je dosiahnuteľný, resp. či je dosiahnuteľný v určenom čase.

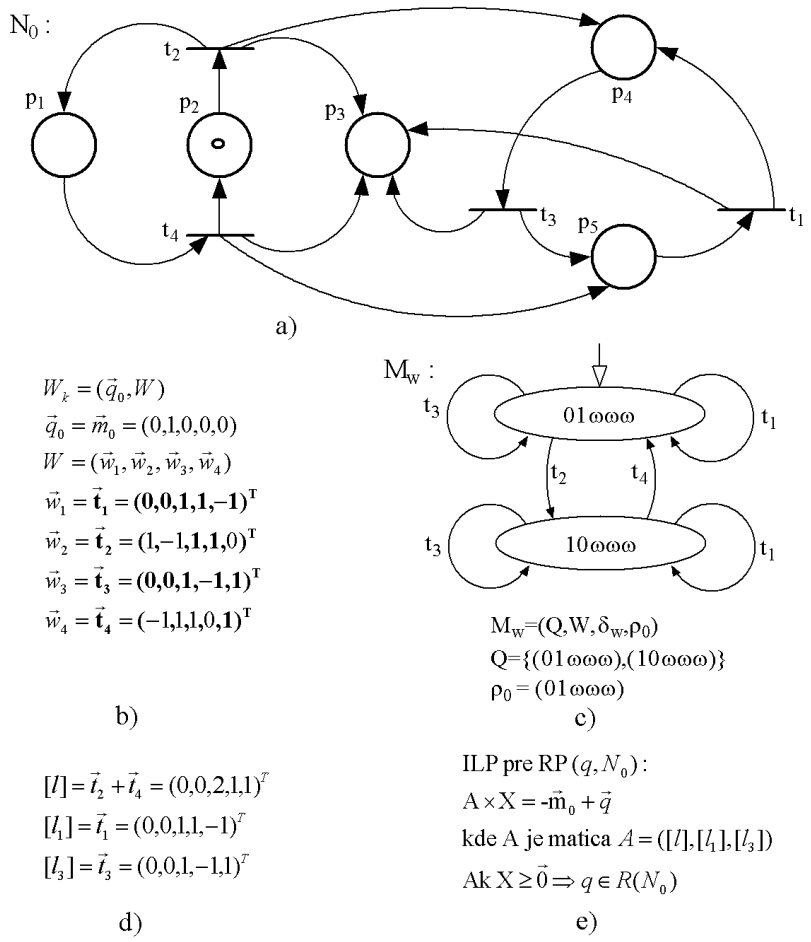
### 2.8.2 Riešenie RP

Od definovania RP bolo urobených mnoho pokusov o jeho riešenie. Jedno originálne riešenie je uvedené v [12]. Pozostáva z nasledujúcich krokov:

1. Reprezentácia PS pomocou VAS (Vektorový Adičný Systém - Vector Addition System).
2. Postupná transformácia stromu dosiahnuteľnosti VAS-u na automat typu $M_w$.
3. Konštrukcia modifikovaného problému celočíselného lineárneho programovania pre daný automat ($MILP_{M_w}$).
4. Riešenie $MILP_{M_w}$.

Pre riešenie RP je vhodné popísať PS pomocou Vektorového adičného systému (VAS). VAS existuje pre každú PS je definovaný takto:

**Definícia 2.8.2.** *$k$-rozmerný VAS je usporiadaná dvojica* $W_k = (q_0, W)$*, kde* $q_0 \in \mathbb{N}^k$ *je počiatočný stav a* $W \subseteq Z^k$ *je konečná množina $k$-rozmerných vektorov. $Z$ je množina celých čísel.*




Obr. 2.5: Príklad riešenia RP: Petriho sieť (a), VAS (b), $M_w$ automat (c), Slučky v automate $M_w$ (d), ILP problém (e)

Pre PS $N_0 = (P, T, pre, post, m_0)$, $|P| = k$, $|T| = n$ zostojíme jej VAS $W_k = (q_0, W)$ tak, že položíme $\vec{m_0} = q_0$ a $W = \{w_0, \ldots, w_n\}$, kde $w_i = \vec{t_i}$.

Podobne, ako PS aj VAS má množinu dosiahnuteľných stavov a je preň definovaný problém dosiahnuteľnosti:

**Definícia 2.8.3.** *Množina dosiahnuteľných stavových vektorov (stavov) VASu* $W_k = (q_0, W)$ *je množina:*

$$R(W_k) = \{q | q \in \mathbb{N}^k, q = q_0 + w_{i_1} + \ldots + w_{i_r}, r \in \mathbb{N},$$

$$\forall j (1 \leq j \leq r) : w_{i_j} \in \mathbb{N}^k, q_j = q_0 + w_{i_1} + \ldots + w_{i_j} \in \mathbb{N}^k\}$$

**Definícia 2.8.4.** *Majme VAS $W_k = (q_0, W)$ a k-rozmerný nezáporný celočíselný vektor q ($k = |P|$). Problém či $q \in R(W_k)$ je nazývaný* problémom dosiahnuteľnosti *pre VAS $W_k$.*

Stavový priestor VASu reprezentuje jeho strom dosiahnuteľnosti - vektorový stavový strom $VST_w$.

**Definícia 2.8.5.** *vektorový stavový strom VASu $W_k = (q_0, W)$ je označený orientovaný strom*

$$VST_W = (T_W, Lab(V), Lab(E), q_0)$$

*kde*

- $T_W = (V, E, r_0)$ *je orientovaný koreňový strom s množinou vrcholov V, množinou hrán $E \subseteq V \times V$ a koreňom $r_0 \subseteq V$,*
- $Lab(V) \subseteq \mathbb{N}^k$ *je množina označení uzlov, definovaná zobrazením $lab_1 : V \to Lab(V)$,*
- $Lab(E) \subseteq W$ *je množina označení hrán, definovaná zobrazením $lab_2 : E \to Lab(V)$.*

Vo všeobecnosti existujú vo $VST_w$ nekonečné cesty. Pomocou trasformácií nekonečných ciest, popísaných v [12], vytvoríme z $VST_w$ konečnú štruktúru - Automat typu $M_w$.

Automat typu $M_w$ je špeciálny konečnostavový automat s interpretáciou. Ide o konečnú štruktúru, ktorá je však schopná popísať nekonečný stavový priestor VASu, resp. Petriho siete. Stavy automatu, nazývané *makrostavy*, reprezentujú stavy (značenia) Petriho siete a prechody medzi stavmi automatu prechody PS. Každý stav $M_w$ je reprezentovaný *stavovým vektorom* $\varrho$, $\varrho \in \mathbb{N}_\omega^k, k = |P|$, kde jednotlivé súradnice predstavujú počet značiek v danom mieste. Stav automatu, ktorého stavový vektor obsahuje $\omega$ reprezentuje nekonečnú podmnožinu stavov siete[9]. Okrem makrostavov sú pre automat $M_w$ charakteristické aj *silne súvislé komponenty*, vytvárajúce slučkovú štruktúru. Každý makrostav je vždy súčasťou slučkovej štruktúry. Namiesto formálneho popisu algoritmu, ktorý je dosť obsiahly, uvedieme príklad (obrázok 2.5). Veľkou výhodou tohto riešenia RP je, že pre danú PS je potrebné zostrojiť $M_w$ automat iba raz a ten istý automat sa použije pre všetky inštancie RP siete. Toto riešenie bolo rozšírené aj pre TRP pre TB siete, kde namiesto $M_w$ automatu konštruujeme obdobným postupom $M_{\tau w}$ automat. Základnými vlastnosťami $M_{\tau w}$ automatu ostáva slučkovitá štruktúra a makrostavy. Množiny

[9]Hodnota $\omega$ predstavuje ľubovoľne veľké prirodzené číslo a množina $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$.

časových značiek (aj nekonečné) sú reprezentované tzv. makrochronosmi. Konkrétny príklad analýzy časovej dosiahnuteľnosti je uvedený v [14].

Okrem popísaného boli navrhnuté aj iné riešenia RP. Napríklad v [4] je navrhnuté riešenie, založené na redukcii RP na problém riešenia polynomickej rovnice.

### 2.8.3 Dekompoziční prístup k riešeniu RP

Ako už bolo spomenuté, zložitosť RP je obrovská a zdá sa, že jedinou možnosťou, ako sa s ňou vysporiadať je použiť pri riešení RP de/kompoziční prístup. V [11] sú navrhnuté 3 spôsoby de/kompozície Petriho siete $N_0 = (P, T, pre, post, m_0)$ na siete $N_1$, $N_2$, $N_i = (P_i, T_i, pre_i, post_i, m_{0_i}), i \in \{1, 2\}$ :

**P-JUNC** dekompozícia je založená na spoločných miestach, ktoré sa vyskytujú v oboch podsieťach: $T_1 \cap T_2 = \varnothing$, $P_1 \cap P_2 = P_c$. Množinu spoločných miest $P_c$ nazývame *množina konformných miest.*

**T-JUNC** dekompozícia je založená na spoločných prechodoch, : $P_1 \cap P_2 = \varnothing$ $T_1 \cap T_2 = T_s$. Množinu $T_S$ nazývame *množina synchrónnych prechodov.*

**PT-JUNC** dekompozícia je založená na spoločných miestach aj prechodoch: $T_1 \cap T_2 = T_s$, $P_1 \cap P_2 = P_c$.

# Kapitola 3

# Jazyk Guarded Commands

Jazyk *Guarded Commands*, tzn. *Jazyk strážených príkazov* (LGC) navrhol E.W. Dijkstra primárne pre svoju publikáciu "A Discipline of Programming" [8], v ktorej sa zaoberá podstatou počítačového programovania, algoritmami a myšlienkovými pochodmi vedúcimi k ich návrhu.

Na systém Dijkstra nazerá ako na nejaký stroj, resp. mechanizmus $S$ ktorý začne svoj výpočet v nejakom počiatočnom stave a skončí vo finálnom stave. Ak je daný mechanizmus *deterministický*, vedie z počiatočného do finálneho stavu jediná cesta. Ak je mechanizmus *nedeterministický*, je takýchto ciest viacero. Stav, resp. skupinu stavov systému možno charakterizovať nejakým predikátom $P$, ktorý tento(tieto) stav(y) spĺňa(jú). Existujú aj dva konštantné predikáty - vždy pravdivý predikát $T$ (true) a vždy nepravdivý predikát $F$ (false).

Pre popis sémantiky svojho jazyka Dijkstra vytvoril tzv. *predikátové transformery*. Predikátový transformer nejakého mechanizmu $S$ je pravidlo, ktorého vstupom je nejaký predikát $R$, definujúci post-podmienku a výstupom predikát *wp(S,R)*, definujúci príslušnú najslabšiu pre-podmienku. Post-podmienka $R$ je predikát, ktorý má platiť po vykonaní $S$. To znamená, že finálny stav je charakterizovaný $R$. O najslabšiej pre-podmienke a jej vlastnostiach pojednáva nasledujúca časť.

## 3.1 Najslabšia pre-podmienka a jej vlastnosti

**Definícia 3.1.1.** *Majme mechanizmus* S *a predikát* R. **Najslabšia pre-podmienka**, **wp(S,R)**, *je predikát (podmienka) charakterizujúci množinu všetkých počiatočných stavov takých, že ak* S *započne výpočet v jednom z týchto počiatočných stavov, tento výpočet skončí (terminuje) v stave spĺňajúcom predikát* R. *Predikát* R *nazývame* post-podmienkou.

Pre wp(S,R) platia štyri základné vlastnosti:

**Veta 3.1.1.** *(Vlastnosť P1 - Zákon vylúčenia zázraku) Pre každý mechanizmus S platí wp(S,F)=F.*

**Veta 3.1.2.** *(Vlastnosť P2 - Monotónnosť) Pre každý mechanizmus S a ľubovoľné post-podmienky $Q, R$ také, že $Q \Rightarrow R$ pre každý stav, taktiež platí, že $wp(S, Q) \Rightarrow wp(S, R)$ pre každý stav.*

**Veta 3.1.3.** *(Vlastnosť P3) Pre každý mechanizmus S a ľubovoľné post-podmienky $Q, R$ platí $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$ pre každý stav.*

**Veta 3.1.4.** *(Vlastnosť P4) Pre každý mechanizmus S a ľubovoľné post-podmienky $Q, R$ platí $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$ pre každý stav.*

Ak je mechanizmus $S$ deterministický platí namiesto vlastnosti *P4* silnejšia vlastnosť *P4'*:

**Veta 3.1.5.** *(Vlastnosť P4') Pre každý deterministický mechanizmus S a ľubovoľné post-podmienky $Q, R$ platí $wp(S, Q) \vee wp(S, R) = wp(S, Q \vee R)$ pre každý stav.*

Prečo Vlastnosť P4' neplatí pre nedeterministické mechanizmy možno ilustrovať na nasledujúcom príklade [8]: Istota, že tehotná žena porodí syna je nulová a istota, že porodí dcéru je tiež nulová, no je isté, že porodí syna alebo dcéru. Teda $wp(S, Q) = wp(S, R) = F$, $wp(S, Q \vee R) = T$ a

$$wp(S, Q \vee R) \Rightarrow (wp(S, Q) \vee wp(S, R)) \; = \; T \Rightarrow F \; = \; F \; .$$

## 3.2 Syntax a sémantika príkazov jazyka

Jazyk strážených príkazov (LGC) pozostáva zo šiestich príkazov. Aj keď ide o "mini" jazyk, ktorý neobsahuje napríklad rekurziu, či procedúry, je postačujúci pre popis algoritmov. V tejto časti si uvedieme syntax konštruktov LGC a ich sémantiku vyjadrenú v zmysle predikátových transformérov[1].

### 3.2.1 Skip a abort

Prvým príkazom LGC je príkaz *skip* (preskočiť), ktorý predstavuje tzv. "prázdny príkaz" - ak sa vykoná, nestane sa nič a systém ostáva v stave, v ktorom bol pred jeho vykonaním. Sémantika *skip* je preto daná najslabšou prepodmienkou v tvare:

$$wp(\text{“}skip\text{”}, R) = R, \text{ pre každú post-podmienku } R \tag{3.1}$$

[1]Na program, zostavený z príkazov LGC možno nazerať, ako na kód pre príslušný predikátový transformér

Príkaz *abort* (zlyhať) má konštantnú najslabšiu pre-podmienku, ktorá vôbec nezávisí od post-podmienky $R$:

$$wp(\text{“}abort\text{”}, R) = F, \text{ pre každú post-podmienku } R \tag{3.2}$$

Príkaz *abort* predstavuje mechanizmus, ktorý, ak je vyvolaný nedokáže sa ani len dostať do finálneho stavu.

### 3.2.2 Priradenie

Príkaz priradenia zapisujeme vo forme $x := E$ a znamená priradenie výrazu $E$ premennej $x$. Operátor := možno čítať, ako “stáva sa”. Sémantika príkazu je definovaná takto:

$$wp(\text{“}x := E\text{”}, R) = R_{E \to x}, \text{ pre každú post-podmienku } R \tag{3.3}$$

Výraz $R_{E \to x}$ je predikát, ktorý vznikne z predikátu $R$ nahradením všetkých výskytov premennej $x$ v $R$ výrazom $E$.

### 3.2.3 Sekvenčná kompozícia

Sekvenčná $S1; S2$ kompozícia predstavuje sekvenčné vykonanie dvoch mechanizmov - najprv sa vykoná $S1$ a potom $S2$. Sémantika kompozicie je daná touto najslabšou pre-podmienkou:

$$wp(\text{“}S1; S2\text{”}, R) = wp(S1, wp(S2, R)), \text{ pre každú post-podmienku } R \tag{3.4}$$

To znamená, že zo stavu spĺňajúceho $wp(\text{“}S1; S2\text{”}, R)$ sa po vykonaní $S1$ dostávame do stavu spĺňajúceho $wp(S2, R)$ a z tohto stavu po vykonaní $S2$ do stavu spĺňajúceho $R$.

### 3.2.4 Príkaz IF

*IF* je príkaz vetvenia, pozostávajúci zo sady *strážených príkazov* $B_i \to SL_i$, $i = 1 \ldots n$, oddelených symbolom “[]”:

if
$B_1 \to SL_1$ []
$B_2 \to SL_2$ []
$\vdots$
$B_n \to SL_n$
fi

Každý *strážený príkaz* pozostáva zo *stráže* $B_i$, čo je vlastne booleovský výraz (podmienka) a *sady príkazov* $SL_i$, ktorá sa môže vykonať len ak je $B_i = T$. Príkaz *IF* sa vykonáva tak, že sa nájde pravdivý strážca $B_i$ a vykoná sa príslušný $SL_i$. Ak je platných viac strážcov súčastne, nedeterministicky sa vyberie jeden z nich a vykoná sa jemu prislúchajúca sada príkazov. Príkaz *IF*, v ktorom sú všetky stráže nepravdivé, je ekvivalentný príkazu *abort*.

Sémantika príkazu je daná najslabšou pre-podmienkou v tvare:

$$\begin{aligned} wp(IF, R) = {} & (\exists j(1 \leq j \leq n) : B_j) \\ & \wedge (\forall j(1 \leq j \leq n) : B_j \Rightarrow wp(SL_j, R)) \end{aligned} \tag{3.5}$$

### 3.2.5 Príkaz DO

Syntax príkazu cyklu *DO* je veľmi podobná *IF* a má tvar:

do
$B_1 \rightarrow SL_1$ []
$B_2 \rightarrow SL_2$ []
$\vdots$
$B_n \rightarrow SL_n$
od

Príkaz *DO* sa vykonáva tak, že sa nájde nejaký pravdivý strážca a vykoná sa príslušná sada príkazov. To sa opakuje, kým je aspoň jeden strážca pravdivý. Ak je platných viac strážcov súčastne, nedeterministicky sa vyberie jeden z nich a vykoná sa jeho sada príkazov. Ak nie je pravdivý ani jeden strážda, vykonávanie *DO* končí. Príkaz *DO*, v ktorom sú hneď na začiatku jeho vykonávania všetky stráže nepravdivé, je ekvivalentný príkazu *skip*.

Pred zadefinovaním $wp(DO, R)$ je potrebné uviesť predikáty $H_0$ a $H_k$. Pod príkazom *IF*, uvedeným v $H_k$ rozumieme príkaz, ktorý dostaneme z príkazu *DO* prepísaním "do" na "if" a "od" na "fi".

$$H_0(R) = R \wedge \neg(\exists j(1 \leq j \leq n) : B_j) \tag{3.6}$$

$$H_k(R) = wp(IF, H_{k-1}(R)) \vee H_0(R) \tag{3.7}$$

$H_k(R)$ je vlastne najslabšia pre-podmienka, že príkaz *DO* skončí (terminuje) po najviac $k$ cykloch a zanechá systém v stave spĺňajúcom post-podmienku $R$. Na základe $H_k(R)$ môžme uviesť najslabšia pre-podmienku pre príkaz *DO*:

$$wp(DO, R) = \exists k(k \geq 0) : H_k(R) \tag{3.8}$$

## 3.3 Konzistencia príkazov LGC s vlastnosťami najslabšej pre-podmienky

V tejto časti uvedieme dôkazy platností vlastností *P1* až *P4* pre jednotlivé príkazy LGC. Prvý konjunt predikátu $wp(IF, R)$ budeme v dôkazoch označovať, ako $BB$:

$$\begin{aligned} BB = &\quad \exists j(1 \leq j \leq n) : B_j \\ wp(IF, R) = &\quad BB \wedge (\forall j(1 \leq j \leq n) : B_j \Rightarrow wp(SL_j, R)) \end{aligned}$$

Taktiež budeme namiesto "$\exists j(1 \leq j \leq n)$" , resp. "$\forall j(1 \leq j \leq n)$" písať iba "$\exists j$", resp. "$\forall j$".

### Pomocné vety

Pred samotnými dôkazmi pre jednotlivé príkazy LGC ešte uvedieme niekoľko platných teorém výrokovej či predikátovej logiky, ktoré budú v dôkazoch požité. Dôkaz týchto pomocných viet ponechávame na čitateľovi.

$$(A \wedge B) \vee (A \wedge C) \;=\; A \wedge (B \vee C) \tag{3.9}$$
$$(A \Rightarrow B) \wedge (A \Rightarrow C) \;=\; A \Rightarrow (B \wedge C) \tag{3.10}$$
$$(A \Rightarrow B) \vee (A \Rightarrow C) \;=\; A \Rightarrow (B \vee C) \tag{3.11}$$
$$(A \wedge B) \;\Rightarrow\; A \tag{3.12}$$

$$\forall j : \Phi(j) \wedge \forall j : \Psi(j) \;=\; \forall j : (\Phi(j) \wedge \Psi(j)) \tag{3.13}$$
$$\forall j : \Phi(j) \vee \forall j : \Psi(j) \;\Rightarrow\; \forall j : (\Phi(j) \vee \Psi(j)) \tag{3.14}$$

V pomocných vetách sú $A$, $B$, $C$, $\Phi(j)$ a $\Psi(j)$ formuly, v $\Phi(j)$ a $\Psi(j)$ sa vyskytuje voľná premenná $j$.

### Príkaz IF

*Dôkaz platnosti vlastnosti P1 (veta 3.1.1) pre IF.* Predpokladáme, že platí:

$$\forall j : wp(SL_j, F) = F \tag{3.15}$$

Ak ani jedna zo stráží $B_1 \ldots B_n$ nie je pravdivá, potom $BB = F$ a vlastnosť *P1* platí.

Ak je aspoň 1 stráž pravdivá, potom $BB = T$ a vlastnosť *P1* bude platiť práve vtedy, ak platí (3.16). (3.16) môžeme, vzhľadom na predpoklad (3.15), prepísať na (3.17).

$$(\forall j : B_j \Rightarrow wp(SL_j, F)) \;=\; F \tag{3.16}$$

$$(\forall j : B_j \Rightarrow F) \;=\; F \tag{3.17}$$

Predikát (3.17) zjavne platí, keďže pre každú pravdivú stráž $B_j$ máme:

$$B_j \Rightarrow F \;=\; T \Rightarrow F \;=\; F$$

□

*Dôkaz platnosti vlastnosti P2 (veta 3.1.2) pre IF.* Predpokladáme, že *P2* platí pre každú sadu príkazov $SL_1$ až $SL_n$ (3.15).

$$(Q \Rightarrow R) \Rightarrow \; \forall j : (wp(SL_j, Q) \Rightarrow wp(SL_j, R)) \tag{3.18}$$

Ak $BB = F$, potom $wp(IF, Q) = wp(IF, R) = F$ a *P2* platí, keďže:

$$(Q \Rightarrow R) \Rightarrow (F \Rightarrow F) \;=\; (Q \Rightarrow R) \Rightarrow T \;=\; T.$$

Ak $BB = T$, potom vzhľadom na predpoklad (3.18) z $Q \Rightarrow R$ vyplýva (3.19).

$$\begin{aligned} wp(IF, Q) \;&=\; \forall j : B_j \Rightarrow wp(SL_j, Q) \\ &\Rightarrow\; \forall j : B_j \Rightarrow wp(SL_j, R) \;= wp(IF, R) \end{aligned} \tag{3.19}$$

To znamená, že $(Q \Rightarrow R) \Rightarrow (wp(IF, Q) \Rightarrow wp(IF, R))$ platí aj ak $BB = T$. V (3.19) sme využili vždy platný predikát (tzn. teorému) (3.20).

$$\begin{aligned} (\forall j : B_j \Rightarrow wp(SL_j, Q) \wedge \forall j : wp(SL_j, Q) \Rightarrow wp(SL_j, R)) \\ \Rightarrow (\forall j : B_j \Rightarrow wp(SL_j, R)) \end{aligned} \tag{3.20}$$

□

*Dôkaz platnosti vlastnosti P3 (veta 3.1.3) pre IF.* Podobne, ako predtým predpokladáme, že *P3* platí pre každú sadu príkazov $SL_1$ až $SL_n$:

$$\forall j : wp(SL_j, Q) \wedge wp(SL_j, R) = wp(SL_j, Q \wedge R) \tag{3.21}$$

Potom:

$$\begin{aligned}
&wp(IF, Q) \wedge wp(IF, R) \\
&= BB \wedge \forall j : B_j \Rightarrow wp(SL_j, Q) \wedge BB \wedge \forall j : B_j \Rightarrow wp(SL_j, R) \\
&= BB \wedge \ \forall j : \big((B_j \Rightarrow wp(SL_j, Q)) \wedge (B_j \Rightarrow wp(SL_j, R))\big) \\
&= BB \wedge \ \forall j : B_j \Rightarrow \big(wp(SL_j, Q) \wedge wp(SL_j, R)\big) \\
&= BB \wedge \ \forall j : B_j \Rightarrow wp(SL_j, Q \wedge R) \\
&= wp(IF, Q \wedge R)
\end{aligned} \tag{3.22}$$

V odvodení (3.22) sme použili pomocné vety (3.13) a (3.10) a predpoklad (3.21).

□

*Dôkaz platnosti vlastnosti P4 (veta 3.1.4) pre IF.* Znova vykonáme dôkaz za predpokladu, že *P4* platí pre všetky sady príkazov v rámci *IF*:

$$\forall j : wp(SL_j, Q) \vee wp(SL_j, R) \Rightarrow wp(SL_j, Q \vee R) \tag{3.23}$$

Potom:

$$\begin{aligned}
&wp(IF, Q) \vee wp(IF, R) \\
&= BB \wedge \ \big((\forall j : B_j \Rightarrow wp(SL_j, Q)) \vee (\forall j : B_j \Rightarrow wp(SL_j, R))\big) \\
&\Rightarrow BB \wedge \ \forall j : \big((B_j \Rightarrow wp(SL_j, Q)) \vee (B_j \Rightarrow wp(SL_j, R))\big) \\
&= BB \wedge \ \forall j : B_j \Rightarrow \big(wp(SL_j, Q) \vee wp(SL_j, R)\big) \\
&\Rightarrow BB \wedge \ \forall j : B_j \Rightarrow wp(SL_j, Q \vee R) \\
&= wp(IF, Q \vee R)
\end{aligned} \tag{3.24}$$

a teda $(wp(IF, Q) \vee wp(IF, R)) \Rightarrow wp(IF, Q \vee R)$. V odvodení (3.24) sme použili pomocné vety (3.9), (3.14) a (3.11) a predpoklad (3.23).

□

### Príkaz DO

Vzhľadom na povahu najslabšej pre-podmienky pre príkaz *DO*, postačí dokázať platnosť vlastností pre $H_k(R)$. Všetky dôkazy sú realizované indukciou podľa $k$ a využívame v nich, že vlastnosti *P1* až *P4* platia aj pre príkaz *IF*.

*Dôkaz platnosti vlastnosti P1 (veta 3.1.1) pre DO.*

***Indukčná báza:*** $H_0(F) = F \wedge \neg BB = F$

***Indukčný krok:***
*Indukčná hypotéza:* $\forall j(j \leq k) : H_j(F) = F$
*Dôkaz pre* $k+1$*:*

$$H_{k+1}(F) = wp(IF, H_k(F)) \vee H_0(F) = wp(IF, F) \vee F = F \vee F = F$$

□

*Dôkaz platnosti vlastnosti P2 (veta 3.1.2) pre DO.*

***Indukčná báza:*** *P2* pre $H_0$ má tvar

$$(Q \Rightarrow R) \;\Rightarrow\; \big((Q \wedge \neg BB) \Rightarrow (R \wedge \neg BB)\big) \tag{3.25}$$

Použitím pomocnej vety (3.12) dostávame (3.26) a z predpokladu vety (3.25) a (3.26) vyplýva (3.27). Podobne ďalším použitím vety (3.12) dostávame (3.28).

$$Q \wedge \neg BB \;\Rightarrow\; Q \tag{3.26}$$
$$Q \wedge \neg BB \;\Rightarrow\; R \tag{3.27}$$
$$Q \wedge \neg BB \;\Rightarrow\; \neg BB \tag{3.28}$$

Použitím pomocnej vety (3.10) na (3.27) a (3.28) dostávame (3.29), čo je záver vety (3.25).

$$(Q \wedge \neg BB) \Rightarrow (R \wedge \neg BB) \tag{3.29}$$

***Indukčný krok:***
*Indukčná hypotéza:* $\forall j(j \leq k) : (Q \Rightarrow R) \Rightarrow (H_j(Q) \Rightarrow H_j(R))$

*Dôkaz pre* $k+1$*:* Za predpokladu, že $Q \Rightarrow R$ dostávame:

$$\begin{array}{ll} & H_{k+1}(Q) \\ = & wp(IF, H_k(Q)) \vee H_0(Q) \\ \Rightarrow & wp(IF, H_k(R)) \vee H_0(Q) \\ \Rightarrow & wp(IF, H_k(R)) \vee H_0(R) \\ = & H_{k+1}(R) \end{array} \tag{3.30}$$

Z (3.30) vyplýva, že ak $Q \Rightarrow R$, potom $H_{k+1}(Q) \Rightarrow H_{k+1}(R)$, čo je vlastne *P2* pre $H_{k+1}$. V (3.30) sme, okrem indukčnej hypotézy, využili aj (3.25) a platnosť *P2* pre *IF*. □

*Dôkaz platnosti vlastnosti P3 (veta 3.1.3) pre DO.*

***Indukčná báza:*** $H_0(Q) \wedge H_0(R) \;=\; Q \wedge R \wedge \neg BB \;=\; H_0(Q \wedge R)$

***Indukčný krok:***
*Indukčná hypotéza:* $\forall j(j \leq k) : (H_j(Q) \wedge H_j(R)) = H_j(Q \wedge R)$

*Dôkaz pre* $k+1$*:* Pre pravú *P3* stranu dostávame:

$$\begin{aligned}
H_{k+1}(Q) \wedge H_{k+1}(R) \quad &= \\
(wp(IF, H_k(Q)) \vee H_0(Q)) \wedge (wp(IF, H_k(R)) \vee H_0(R)) \quad &= \\
\Big(\big(wp(IF, H_k(Q)) \wedge wp(IF, H_k(R))\big) \vee \big(H_0(Q) \wedge H_0(R)\big)\Big) \quad &(3.31)\\
\vee \big(wp(IF, H_k(Q)) \wedge H_0(R)\big) \quad &(3.32)\\
\vee \big(wp(IF, H_k(R)) \wedge H_0(Q)\big) \quad &(3.33)
\end{aligned}$$

Z toho sa (3.32) redukuje na $F$ (*false*), keďže

$$\begin{aligned}
&wp(IF, H_k(Q)) \wedge H_0(R)\\
&= BB \wedge \big(\forall i(1 \leq i \leq n) : B_i \Rightarrow wp(SL_i, H_k(Q))\big) \wedge R \wedge \neg BB\\
&= F
\end{aligned}$$

Analogicky sa na $F$ redukuje aj (3.33). Použijúc platnosť *P3* pre *IF*, indukčnú hypotézu a bázu na zostávajúcu časť (3.31) nakoniec dostávame:

$$H_{k+1}(Q) \wedge H_{k+1}(R) = \; wp(IF, H_k(Q \wedge R)) \vee H_0(Q \wedge R) = H_{k+1}(Q \wedge R)$$

□

Dôkaz platnosti vlastnosti *P4* pre *DO* je veľmi podobný dôkazu vlastnosti *P3* a nebudeme ho tu uvádzať.

## 3.4 Vlastnosti príkazov LGC

**Veta 3.4.1.** *(Základný teorém pre IF) Ak* $Q \Rightarrow BB$ *a* $\forall j(1 \leq j \leq n)$ : $Q \wedge B_j \Rightarrow wp(SL_j, R)$ *potom* $Q \Rightarrow wp(IF, R)$

**Veta 3.4.2.** *(Základný teorém pre DO) Ak* $(P \wedge BB) \Rightarrow wp(IF, P)$ *potom* $(P \wedge wp(DO, T)) \Rightarrow wp(DO, P \wedge \neg BB)$

# Kapitola 4

# B-Metóda

*B-Metóda*[1] a jej špecifikačný a návrhový jazyk *B - notácia abstraktných strojov* (B - Abstract Machine Notation, *B-AMN*) boli vytvorené v rokoch 1985 až 1988 J.R. Abrialom, výskumnou skupinou "Programming Research Group" na Oxfordskej univerzite a výskumným oddelením firmy British Petroleum International, ktorá projekt iniciovala a sponzorovala.

B-AMN vychádza z formálneho špecifikačného jazyka Z (Z notation), ktorého autorstvo tiež prislúcha najmä J.R. Abrialovi. B-Metóda podporuje nielen formálnu špecifikáciu systému, ale aj celý vývojový proces od formálnej špecifikácie cez sériu zjemnení až po spustiteľnú implementáciu vo zvolenom programovacom jazyku. Pomocou tzv. *povinných dôkazov* (Proof Obligations - *POb*) je možné overiť vnútornú konzistenciu špecifikácie a taktiež verifikovať zjemnenie voči abstraktnejšej špecifikácii. Za účelom validácie formálnej špecifikácie voči požiadavkám zákazníka (ktoré sú popísané abstraktne, ale neformálne) je možné použiť animačné techniky.

Pre návrh a vývoj v B existujú dva robustné komerčne dostupné nástroje *B-Toolkit* a *Atelier B*. B-Toolkit je vyvíjaný firmou B-Core [33]. Tá síce od roku 1993 prevzala patronát nad ďalším vývojom B-metódy, no veľmi aktívna v tomto smere nie je[2]. Atelier B [31] bol vytvorený francúzskou firmou Digilog v spolupráci s J.R. Abrialom. Po Digilog-u vývoj a podporu nástroja Atelier B prebrala firma ClearSy [34]. ClearSy s Atelier B je v súčasnosti (2008) zrejme jediným väčším hráčom na poli komerčného využitia B-Metódy a podieľa sa aj na jej ďalšom vývoji. B-Metóda je v poslednej dobe využívaná najmä na vývoj riadiaceho softvéru pre automatizované linky metra.

Stručný popis B-Metódy uvedený v tejto kapitole vychádza najmä z publikácie [22], ktorá sa problematike použitia B-Metódy pri návrhu diskrétnych systémov venuje hlbšie a je v nej uvedených množstvo príkladov, a z pub-

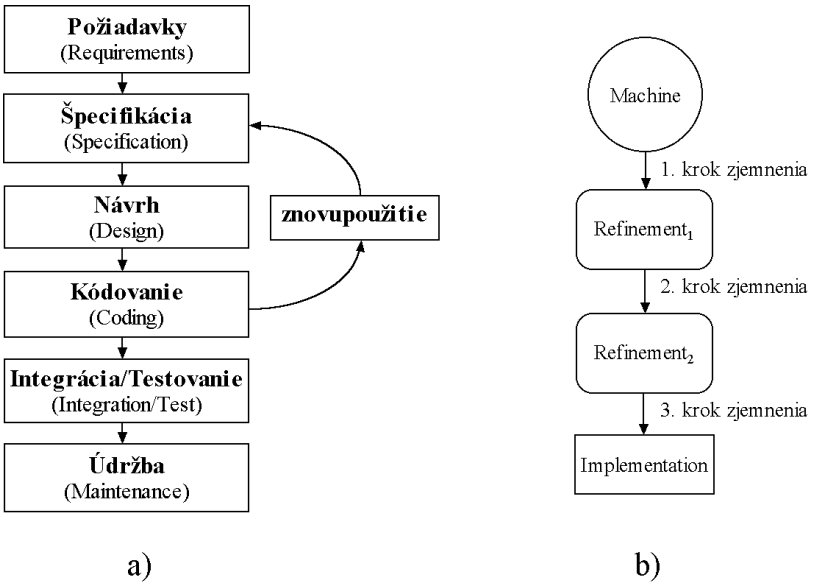
[1] Namiesto pojmu "B-Metóda" budeme tiež používať iba písmeno "B".

[2] O tom svedčí aj fakt, že jej web stránka nebola aktualizovaná od roku 2002

likácie [1], ktorú možno považovať za základnú literatúru o B-Metóde a jej jazyku. Dobrým zdrojom informácií o B-Metóde sú tiež web stránky firmy B-Core [33] a nástroja *B4free* [32], čo je oklieštená bezplatná verzia nástroja Atelier B.

## 4.1 Vývojový proces v B

Úplný proces vývoja systému v B-Metóde je znázornený na obrázku 4.1 a).




Obr. 4.1: Vývoj v B: životný cyklus vývoja v B (a) a príklad sekvencie špecifikačných komponentov v B (b)

Tento proces, reps. životný cyklus, vývoja systému pozostáva z nasledujúcich fáz [22]:

1. **Analýza požiadaviek:**
   Vytvorenie neformálneho alebo štruktúrovaného modelu problémovej domény a požiadaviek na vytváraný systém. Výsledkom je *množina analytických modelov* (napríklad UML diagramy).

2. **Vývoj špecifikácie:**

   (a) Formalizácia prvkov analytických modelov do podoby abstraktných strojov. Analytické modely sa pritom používajú aj na dekompozíciu špecifikácie na koncepčne zmysluplné komponenty.

(b) Animácia na overenie (validáciu) špecifikácie voči vybraným požiadavkám na systém a testovacím scenárom.

(c) Generovanie podmienok vnútornej konzistencie a ich dôkaz. Tieto podmienky nazývame *povinné dôkazy* abstraktného stroja a sú popísané v časti 4.4.2.

Výsledkom tejto fázy je formálna *abstraktná špecifikácia systému*. Tá v B pozostáva z kolekcie špecifikačných komponentov nazývaných *abstraktné stroje* (MACHINEs), navzájom poprepájaných (sprístupnených) kompozičnými mechanizmami, ktoré B-Metóda poskytuje (kapitola 4.4).

3. **Návrh**

(a) Identifikácia dekompozície implementácie systému, zahŕňajúcej znovupoužiteľné komponenty z predošlého vývoja alebo špecifikačných knižníc.

(b) Zjemnenie vybraných komponentov formálnej špecifikácie. Postupným zjemňovaním komponentu vzniká sekvencia špecifikácií komponentov typu *zjemnenie* (REFINEMENT), ktorej konečným výsledkom je špecifikačný komponent *implementácia* ( IMPLEMENTATION). Proces zjemňovania ilustruje obrázok obrázok 4.1 b.

(c) Generovanie povinných dôkazov vytvorených zjemnení a ich dôkaz. Povinné dôkazy zjemnenia overujú, či zjemnenie konkretizuje (zjemňuje) príslušný abstraktnejší komponent tak, ako je to dané v invariante zjemnenia.

Produktom fázy je konkrétny *formálny návrh*, pozostávajúci z jedného, alebo viacerých špecifikačných komponentov IMPLEMENTATION.

4. **Kódovanie, integrácia, testovanie**

(a) Použitie generátora kódu na návrhy najnižšej úrovne - na implementácie.

(b) Testovanie vygenerovaného kódu s použitím modelových prípadov založených na požiadavkách systému.

Táto fáza produkuje *vykonateľnú implementáciu* v konkrétnom programovacom jazyku (napr. ADA, C či Java).

Z uvedených krokov sú samotnou B-Metódou a nástrojmi pre ňu vytvorenými podporované kroky 2 až 4a.

## 4.2 Predikáty a výrazy v B-AMN

Jazyk B-AMN, tiež označovaný ako *B jazyk* (*B language*) [30], je možné rozdeliť na dve časti:

- **jazyk pre zápis výrazov a predikátov** umožňuje zapísať výrazy a predikáty definujúce parametre, konštanty, množiny, premenné a invariantné vlastnosti špecifikačného komponentu. Predikáty a výrazy sú použité aj v rámci konštruktov GSL. Jazyk je vybudovaný na základoch klasickej logiky a teórie množín (Zermelo-Frankel).
- **jazyk zovšeobecnenej substitúcie (GSL)** obsahuje konštrukty (príkazy), pomocou ktorých sú definované operácie špecifikačného komponentu. Je mu venovaná časť 4.3.

Ďalej je uvedený podrobnejší popis predikátov a väčšiny[3] výrazov, pričom sú rozdelené podľa toho s akými typmi matematických objektov manipulujú. Kompletný popis je možné nájsť v referenčnom manuáli B-AMN pre B-Toolkit [29], kde prípustné matematické objekty sú prirodzené čísla, usporiadané dvojice, množiny, relácie funkcie a sekvencie. Jazyk B-AMN pre Atelier B [30] je bohatší ako pre B-Toolkit, umožňuje napríklad používať aj celé čísla, záznamy a stromy.

### 4.2.1 Všeobecné predikáty a výrazy

Všeobecné predikáty a (jediný) všeobecný výraz sú uvedené v tabuľkách 4.1 a 4.2.

Každá tabuľka je rozdelená na 3 stĺpce. V prvom ("Výraz", resp. "Predikát") je uvedený matematický zápis výrazu, resp. predikátu, v druhom stĺpci ("ASCII") je jeho zápis používaný pri písaní špecifikácie daného komponentu[4] a v treťom ("Význam") jeho význam. V tabuľkách 4.1 a 4.2 sú $P$, $Q$ predikáty, $E$, $F$ výrazy, $S$, $T$ množiny, $z$ je zoznam premenných a $x$ je premenná.

Pod symbolom $\mathcal{P}_0(x)$ budeme v tabuľkách 4.1 až 4.8 rozumieť obmedzujúci predikát v tvare $x \in S$, $x = E$, $x \subset S$ alebo $x \subseteq S$, kde $z \backslash S$ a $z \backslash E$. Zápis $z \backslash E$ znamená, že vo výraze $E$ nie je žiaden voľný výskyt premenných zo $z$.

[3] Sú vynechané niektoré redundantné výrazy pre množiny, relácie a sekvencie, ktoré môžu byť skonštruované pomocou iných, tu uvedených, výrazov.

[4] Špecifikačné komponenty sú písané vo forme textových súborov, kde nie je možné priamo zapisovať matematické symboly.

| **Predikát** | **ASCII** | **Význam** |
|---|---|---|
| $P \wedge Q$ | `P & Q` | Konjunkcia: "$P$ a $Q$". |
| $P \vee Q$ | `P or Q` | Disjunkcia: "$P$ alebo $Q$". |
| $P \Rightarrow Q$ | `P => Q` | Implikácia: "ak $P$ potom $Q$". |
| $P \Leftrightarrow Q$ | `P <=> Q` | Ekvivalencia: "$P$ práve vtedy ak $Q$". |
| $\neg P$ | `not(P)` | Negácia: "nie $P$". |
| $true$ | `true` | Vždy pravdivý predikát. |
| $\forall z.(Q \Rightarrow P)$ | `!(z).(Q => P)` | Všeobecná kvantifikácia: "pre každé $z$, ak platí $Q$, potom platí $P$". Predikát $Q$ musí pre každú premennú $x$ zo zoznamu $z$ obsahovať $\mathcal{P}_0(x)$. |
| $\exists z.P$ | `#(z).P` | Existenčná kvantifikácia: "pre niektoré $z$ platí $P$". $P$ musí pre každú $x$ zo $z$ obsahovať $\mathcal{P}_0(x)$. |
| $E = F$ | `E = F` | Rovnosť: "$E$ je rovný $F$". |
| $E \neq F$ | `E /= F` | Nerovnosť: "$E$ je rôzny od $F$". |

Tabuľka 4.1: Všeobecné predikáty

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $E \mapsto F$ | `E \|-> F` | Usporiadaná dvojica $(E, F)$. |

Tabuľka 4.2: Všeobecný výraz

### 4.2.2 Množinové predikáty a výrazy

Predikáty a niektoré výrazy pre množiny sú uvedené v tabuľkách 4.3 a 4.4. v tabuľkách sú použité rovnaké symboly, ako v časti 4.2.1 a navyše symbol $R$ označujúci množinu. Každá množina je zároveň výrazom.

| **Predikát** | **ASCII** | **Význam** |
|---|---|---|
| $E \in S$ | `E: S` | Príslušnosť k množine : "$E$ patrí do $S$". |
| $E \notin S$ | `E /: S` | "$E$ nepatrí do $S$." |
| $S \subseteq T$ | `S <: T` | "$S$ je podmnožina $T$."<br>$S \subseteq T \equiv \forall x.(x \in S \Rightarrow x \in T)$ |
| $S \not\subseteq T$ | `S /<: T` | "$S$ nie je podmnožina $T$."<br>$S \not\subseteq T \equiv \neg(S \subseteq T)$ |
| $S \subset T$ | `S <<: T` | "$S$ je vlastná podmnožina $T$."<br>$S \subset T \equiv (S \subseteq T) \wedge \neg(T \subseteq S)$ |
| $(S \not\subset T)$ | `S /<<: T` | "$S$ nie je vlastná podmnožina $T$."<br>$S \not\subset T \equiv \neg(S \subset T)$ |

Tabuľka 4.3: Množinové predikáty

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $\varnothing$ | `{}` | prázdna množina |
| $\{z\|P\}$ | `{z\|P}` | Definovanie množiny pomocou vlastností jej prvkov (set comprehension). Predikát $P$ musí pre každú $x$ zo $z$ obsahovať $\mathcal{P}_0(x)$. |
| $\{z\|z \in R \wedge P\}$ | `{z\|z:R & P}` | Iný zápis set comprehension. |
| $\{E\}$ | `{E}` | Množina s jediným prvkom, definovaným $E$. |
| $\{E, \ldots, F\}$ | `{E,...,F}` | Množina s prvkami, definovanými $E$ až $F$. |
| $S \times T$ | `S*T` | Kartézsky súčin množín $S$ a $T$.<br>$S \times T = \{(x, y)\|\ x \in S \wedge y \in T\}$ |
| $\mathbb{P}(S)$ | `POW(S)` | Množina všetkých podmnožín (power set) $S$.<br>$(x \in \mathbb{P}(S)) \Leftrightarrow (x \subseteq S)$ |
| $\mathbb{P}_1(S)$ | `POW1(S)` | Množina všetkých neprázdnych podmnožín $S$.<br>$\mathbb{P}_1(S) = \mathbb{P}(S) - \varnothing$ |
| $\mathbb{F}(S)$ | `FIN(S)` | Množina všetkých konečných podmnožín $S$. |
| $\mathbb{F}_1(S)$ | `FIN1(S)` | Množina všetkých neprázdnych konečných podmnožín $S$. |
| $S \cup T$ | `S \/ T` | Zjednotenie množín $S$ a $T$.<br>$S \cup T = \{x\|\ x \in S \vee x \in T\}$ |
| $S \cap T$ | `S /\ T` | Prienik množín $S$ a $T$.<br>$S \cap T = \{x\|\ x \in S \wedge x \in T\}$ |
| $S - T$ | `S - T` | Rozdiel množín $S$ a $T$.<br>$S - T = \{x\|\ x \in S \wedge x \notin T\}$ |

Tabuľka 4.4: Niektoré množinové výrazy

### 4.2.3 Predikáty a výrazy pre prirodzené čísla

Prirodzené čísla, teda nezáporné celé čísla, sú tiež výrazmi. Predikáty a výrazy pre ne sú uvedené v tabuľkách 4.5 a 4.6, kde sú použité rovnaké symboly ako v časti 4.2.1 a tiež symboly $n$ a $m$ pre prirodzené čísla, resp. výrazy pre prirodzené čísla.

| **Predikát** | **ASCII** | **Význam** |
|---|---|---|
| $m > n$ | `m > n` | " $m$ je väčšie ako $n$." |
| $m < n$ | `m < n` | " $m$ je menšie ako $n$." |
| $m \geq n$ | `m >= n` | " $m$ je väčšie ako alebo rovné $n$." |
| $m \leq n$ | `m <= n` | " $m$ je menšie ako alebo rovné $n$." |

Tabuľka 4.5: Predikáty pre prirodzené čísla

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $\mathbb{N}$ | `NAT`, `NATURAL` | Množina prirodzených čísel. |
| $\mathbb{N}_1$ | `NAT1`, `NATURAL1` | Množina nenulových prirodzených čísel. |
| $min(S)$ | `min(S)` | Minimum z množiny $S$, $S \in \mathbb{P}_1(\mathbb{N})$. |
| $max(S)$ | `max(S)` | Maximum z množiny $S$, $S \in \mathbb{F}_1(\mathbb{N})$. |
| $m + n$ | `m + n` | Súčet $m$ a $n$. |
| $m - n$ | `m - n` | Rozdiel $m$ a $n$ (definované pre $m \geq n$). |
| $m \times n$ | `m * n` | Súčin $m$ a $n$. |
| $m/n$ | `m / n` | Celočíselné delenie $m$ číslom $n$. |
| $m \; mod \; n$ | `m mod n` | Zvyšok po celočíselnom delení $m/n$. |
| $m..n$ | `m..n` | Množina prirodzených čísel od $m$ po $n$. |
| $card(S)$ | `card(S)` | Počet prvkov konečnej množiny $S$. |
| $\Sigma z.(P\|F)$ | `SIGMA(z).(P\|F)` | Súčet hodnôt výrazu $F$ nad prirodzenými číslami pre $z$ také, že $P$ platí. $P$ musí pre každú $x$ zo $z$ obsahovať $\mathcal{P}_0(x)$. |
| $\Pi z.(P\|F)$ | `PI(z).(P\|F)` | Súčin hodnôt výrazu $F$ nad prirodzenými číslami pre $z$ také, že $P$ platí. $P$ musí pre každú $x$ zo $z$ obsahovať $\mathcal{P}_0(x)$. |

Tabuľka 4.6: Výrazy pre prirodzené čísla

V jazyku B-AMN "*francúzkej školy B*" (Atelier B) je dovolené používať aj celé čísla (množina celých čísel sa označuje $\mathbb{Z}$). Pre celé čísla sú použiteľné tie isté predikáty a výrazy, ako pre prirodzené čísla, s tým rozdielom, že výraz

$m - n$ je definovaný pre každé $m, n$. Rozdiel je taktiež v označení množiny prirodzených čísel: B-AMN pre B-Toolkit [29] pozná iba označenie `NAT`, resp. `NAT1`, zatiaľ čo v jazyku Atelier B [30] sa množina prirodzených čísel označuje ako `NATURAL` a `NAT` je implementovateľná množina prirodzených čísel (tzn. že má horné ohraničenie). Analogicky sa rozlišuje `NAT1` a `NATURAL1`.

### 4.2.4 Výrazy pre relácie

Reláciou je každá podmnožina nejakého kartézskeho súčinu. Relácia je teda množina usporiadaných dvojíc a je tiež výrazom. Niektoré výrazy pre relácie sú uvedené v tabuľke 4.7, kde $S$, $T$, $U$, $V$ sú množiny, $r$, $p$, $q$ sú relácie, $s \subseteq S$ a $t \subseteq T$.

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $S \leftrightarrow T$ | `S <-> T` | Množina relácií z $S$ do $T$.<br>$S \leftrightarrow T = \mathbb{P}(S \times T)$ |
| $dom(r)$ | `dom(r)` | Definičný obor relácie $r$, $r \in S \leftrightarrow T$.<br>$dom(r) = \{x \mid x \in S \wedge \exists y.(y \in T \wedge (x,y) \in r)\}$ |
| $ran(r)$ | `dom(r)` | Obor hodnôt relácie $r$, $r \in S \leftrightarrow T$.<br>$ran(r) = \{y \mid y \in T \wedge \exists x.(x \in S \wedge (x,y) \in r)\}$ |
| $p;q$ | `p;q` | Kompozícia relácií $p \in S \leftrightarrow T$ a $q \in T \leftrightarrow U$.<br>$p;q = \{(x,z) \mid (x,z) \in S \times U \wedge$<br>$\exists y.(y \in T \wedge (x,y) \in p \wedge (y,z) \in q)\}$ |
| $p \circ q$ | `p circ q` | $p \circ q = q;p$ |
| $id(S)$ | `id(S)` | $id(S) = \{(x,y) \mid (x,y) \in S \times S \wedge x = y\}$ |
| $s \lhd r$ | `s <\| r` | Zúženie $r$, $r \in S \leftrightarrow T$, podľa $s$.<br>$s \lhd r = \{(x,y) \mid (x,y) \in r \wedge x \in s\}$ |
| $r \rhd t$ | `r \|> t` | $r \rhd t = \{(x,y) \mid (x,y) \in r \wedge y \in t\}$ |
| $r^{-1}$ | `r~` | Inverzná relácia k $r$, $r \in S \leftrightarrow T$.<br>$r^{-1} = \{(y,x) \mid (y,x) \in T \times S \wedge (x,y) \in r\}$ |
| $p \otimes q$ | `p >< q` | Priamy súčin $p \in S \leftrightarrow U$ a $q \in S \leftrightarrow V$.<br>$p \otimes q = \{(x,(y,z)) \mid (x,(y,z)) \in S \times (U \times V)$<br>$\wedge (x,y) \in p \wedge (x,z) \in q\}$ |
| $p \parallel q$ | `p \|\| q` | Paralelný súčin $p \in S \leftrightarrow T$ a $q \in V \leftrightarrow U$.<br>$p \parallel q = \{((x,y),(m,n)) \mid ((x,y),(m,n)) \in$<br>$(S \times V) \times (T \times U) \wedge (x,m) \in p \wedge (y,n) \in q\}$ |
| $r^n$ | `iterate(r,n)` | $n$-tá iterácia $r$, $n \in \mathbb{N}$, $r \in S \leftrightarrow S$.<br>$r^0 = id(S),\ r^{n+1} = r;r^n$ |

Tabuľka 4.7: Niektoré výrazy pre relácie

### 4.2.5 Výrazy pre funkcie

Funkcie, alebo zobrazenia, sú relácie, v ktorých platí, že každý prvok z definičného oboru je vo vzťahu (relácii) s jedinečným prvkom z oboru hodnôt. Funkcia teda nemôže obsahovať dve rôzne dvojice s rovnakým prvým prvkom. Funkcie sú takisto výrazmi. Výrazy pre funkcie sú uvedené v tabuľke 4.8, kde $P$ je predikát, $E$ výraz, $S$, $T$ sú množiny a $z$ je zoznam premenných.

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $S \nrightarrow T$ | `S +-> T` | Množina parciálnych funkcií z $S$ do $T$. $S \nrightarrow T = \{r \mid r \in S \leftrightarrow T \wedge (r^{-1}; r) \subseteq id(T)\}$ |
| $S \rightarrow T$ | `S --> T` | Množina úplných funkcií z $S$ do $T$. $S \rightarrow T = \{f \mid f \in S \nrightarrow T \wedge dom(f) = S\}$ |
| $S \rightarrowtail T$ | `S >+> T` | Množina parciálnych injektívnych funkcií z $S$ do $T$. $S \rightarrowtail T = \{f \mid f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S\}$ |
| $S \rightarrowtail T$ | `S >-> T` | Množina úplných injektívnych funkcií z $S$ do $T$. $S \rightarrowtail T = (S \rightarrowtail T) \cap (S \rightarrow T)$ |
| $S \twoheadrightarrow T$ | `S +->> T` | Množina parciálnych surjektívnych funkcií z $S$ do $T$. $S \twoheadrightarrow T = \{f \mid f \in S \nrightarrow T \wedge ran(f) = T\}$ |
| $S \twoheadrightarrow T$ | `S -->> T` | Množina úplných surjektívnych funkcií z $S$ do $T$. $S \twoheadrightarrow T = (S \twoheadrightarrow T) \cap (S \rightarrow T)$ |
| $S \rightarrowtail\!\!\!\!\twoheadrightarrow T$ | `S >->> T` | Množina bijektívnych funkcií z $S$ do $T$. $S \rightarrowtail\!\!\!\!\twoheadrightarrow T = (S \twoheadrightarrow T) \cap (S \rightarrowtail T)$ |
| $\lambda z.(z \in S \wedge P \mid E)$ | `%(z).(z: S & P \| E)` | Konštrukcia funkcie. Je to funkcia $\{(z, y) \mid z \in S \wedge y \in E \wedge P\}$, s doménou $\{z \mid z \in S \wedge P\}$, pričom $y \backslash E$ a $y \backslash P$. |
| $\lambda z.(P \mid E)$ | `%(z).(P \| E)` | Konštrukcia funkcie. Predikát $P$ musí pre každú $x$ zo $z$ obsahovať $\mathcal{P}_0(x)$. |
| $f(x)$ | `f(x)` | Hodnota funkcie $f$ v $x$, $x \in dom(f)$. |

Tabuľka 4.8: Výrazy pre funkcie

Tvar predikátu $\mathcal{P}_0(x)$ je definovaný v časti 4.2.1.

### 4.2.6 Výrazy pre sekvencie

Sekvencia nad množinou $S$ je parciálna funkcia z $\mathbb{N}$ do $S$, ktorej definičný obor je interval $1..n$, $n \in N$. Sekvencie sú takisto výrazmi. Niektoré výrazy pre sekvencie sú uvedené v tabuľke 4.9, kde $E$, $F$ sú výrazy, $S$ je množina, $s$, $t$ sú sekvencie prvkov z $S$ a $e$ je prvok z $S$.

| **Výraz** | **ASCII** | **Význam** |
|---|---|---|
| $[\,]$ | `<>` | Prázdna sekvencia. |
| $[E]$ | `[E]` | Sekvencia s jediným prvkom, definovaným $E$. $[E] = \{1 \rightarrow E\}$ |
| $[E, \ldots, F]$ | `[E,...,F]` | Sekvencia s prvkami, definovanými $E$ až $F$. |
| $size(s)$ | `size(s)` | Dĺžka konečnej sekvencie $s$. |
| $seq(S)$ | `seq(S)` | Množina konečných sekvencií prvkov z $S$. |
| $seq_1(S)$ | `seq1(S)` | Množina konečných neprázdnych sekvencií prvkov z $S$. |
| $iseq(S)$ | `iseq(S)` | Množina injektívnych sekvencií prvkov z $S$. $iseq(S) = seq(S) \cap (\mathbb{N}_1 \rightarrowtail S)$. |
| $perm(S)$ | `perm(S)` | Množina permutácií $S$. $perm(S) = (1..card(S)) \rightarrowtail\!\!\!\!\!\rightarrow S$. |
| $s^\frown t$ | `s^t` | Zreťazenie sekvencií $s$ a $t$. |
| $e \rightarrow e$ | `e -> s` | Sekvencia vytvorená pridaním $e$ na začiatok $s$. |
| $s \leftarrow e$ | `s <- e` | Sekvencia vytvorená pridaním $e$ na koniec $s$. |
| $rev(s)$ | `rev(s)` | Sekvencia s opačným poradím prvkov, ako $s$. |
| $s \uparrow n$ | `s /\| n` | Sekvencia vytvorená z $s$ ponechaním iba jej prvých $n$ prvkov, $n \leq size(s)$. |
| $s \downarrow n$ | `s \\/ n` | Sekvencia vytvorená z $s$ odstránením jej prvých $n$ prvkov, $n \leq size(s)$. |
| $first(s)$ | `first(s)` | Prvý prvok neprázdnej sekvencie $s$. |
| $last(s)$ | `last(s)` | Posledný prvok neprázdnej sekvencie $s$. |
| $tail(s)$ | `tail(s)` | Sekvencia vytvorená odstránením prvého prvku z neprázdnej sekvencie $s$. |
| $front(s)$ | `front(s)` | Sekvencia vytvorená odstránením posledného prvku z neprázdnej sekvencie $s$. |

Tabuľka 4.9: Niektoré výrazy pre sekvencie

## 4.3 Zovšeobecnená substitúcia

Operácie špecifikačných komponentov v B-Metóde sú zapísané v *jazyku zovšeobecnenej substitúcie* (Generalized Substitution Language, GSL), ktorý je súčasťou B-AMN. Konštrukty GSL sa nazývajú *zovšeobecnené substitúcie* (Generalized Substitution, GS). Jazyk GSL bol navrhnutý tak, aby

- zjednodušoval požiadavky na dôkazy,
- poskytol jednotný zápis od abstraktných strojov až k procedurálnemu kódu v implementácii,
- podporoval dekompozíciu operácií v zjemneniach abstraktného stroja.

### 4.3.1 Syntax zovšeobecnej substitúcie

Prehľad konštruktov (zovšeobecnených substitúcií) jazyka GSL a ich intuitívny význam je v tabuľke 4.10. Symboly použité v tabuľkách 4.10 až 4.13 majú takýto význam:

- $S$, $S_1$, $S_2$, $T$, $U$ sú zovšeobecnené substitúcie,
- $I$, $E$, $P$, $Q$ sú predikáty,
- $x$, $v$ sú premenné alebo zoznamy premenných,
- $e$, $ve$ sú zoznamy výrazov nad premennými; $e$ má rovnaký rozmer ako $x$,
- $xx$, $vv$ sú premenné.

Rozdiel medzi $P|S$ a $P \Longrightarrow S$ je možné intuitívne pochopiť nasledovne: v $P|S$ by sa vykonávanie $S$ nemalo vyvolať ak $P$ nie je splnené a v $P \Longrightarrow S$ sa vykonávanie $S$ nemôže vyvolať ak $P$ nie je splnené. Ak platí $P$, potom je vykonanie $P|S$ aj $P \Longrightarrow S$ rovnaké: vykoná sa GS $S$. Ak $P$ neplatí, tak vykonanie $P|S$ môže viesť do ľubovoľného stavu a nemusí ani terminovať[5]. Na druhej strane GS $P \Longrightarrow S$ je pri neplatnosti $P$ nevykonateľná.

Niektoré GS, alebo ich kombinácie sa v operáciách špecifikačných komponentov zapisujú v pozmenenom tvare, ako to ukazuje tabuľka 4.11. Konštrukty ANY a VAR zavádzajú novú lokálnu premennú $v$, ktorá v prípade ANY je obmedzená splnením predikátu $P$. Konštrukt ANY nazývame *zovšeobecnený strážený príkaz*. V rôznych fázach vývoja systému je možné používať iba podmnožinu týchto substitúcií:

[5] V prípade neplatnosti $P$ je $P|S$ definovaná rovnako ako príkaz *Abort* v jazyku strážených príkazov E.W. Dijkstru [8].

| | |
|---|---|
| $x := e$ | jednoduché priradenie (základná substitúcia) |
| SKIP | prázdna GS: nerob nič |
| $S_1 [] S_2$ | viazaná voľba : rob $S_1$ alebo $S_2$ |
| $P \mid S$ | pre-podmienka : ak $P$ platí, správaj sa ako $S$ |
| $P \Longrightarrow S$ | strážený príkaz : iba ak $P$ platí, vykonaj $S$ |
| $@v.S$ | neviazaný nedeterminizmus:<br>pre nejaké $v$ vykonaj $S$ |
| $S_1; S_2$ | sekvenčná kompozícia : vykonaj $S_1$, potom $S_2$ |
| $S_1 \parallel S_2$ | viacnásobná GS : vykonaj $S_1$ a $S_2$ |
| WHILE $E$ DO $S$<br>INVARIANT $I$<br>VARIANT $ve$ END | cyklus: rob $S$ kým platí $E$ |

Tabuľka 4.10: Syntax GS

| | |
|---|---|
| $S_1 [] S_2$ | CHOICE $S_1$ OR $S_2$ END |
| $P \mid S$ | PRE $P$ THEN $S$ END |
| $P \Longrightarrow S$ | SELECT $P$ THEN $S$ END |
| $@v.S$ | VAR $v$ IN $S$ END |
| $@v.(P \Longrightarrow S)$ | ANY $v$ WHERE $P$ THEN $S$ END |
| $(E \Longrightarrow S_1) [] (\neg E \Longrightarrow S_2)$ | IF $E$ THEN $S_1$ ELSE $S_2$ END |
| ANY $vv$ WHERE $vv \in ss$<br>THEN $xx := vv$ END | $xx :\in ss$ |

Tabuľka 4.11: Zápis niektorých GS v operáciách strojov

- Vo fáze abstraktnej špecifikácie systému, tzn. v abstraktných strojoch, nie je povolené používať sekvenčnú kompozíciu (;), WHILE cykly a konštrukt VAR.
- V zjemnenich je povolené používať všetky substitúcie okrem konštruktov WHILE a VAR.
- V záverečnej, implementačnej, fáze (tzn. v implementáciách) je možné používať iba procedurálne konštrukty, ako priradenie (v:=e), cyklus WHILE, príkaz vetvenia IF - THEN - ELSE a príkaz VAR.

Pre uľahčenie písania špecifikácií poskytuje jazyk GSL aj ďalšie konštrukty. Tie sú však len alternatívnym zápisom niektorých kombinácií tu uvedených zovšeobecnených substitúcií. Úplný prehľad konštruktov GSL ako aj celého B-AMN možno nájsť v [29].

### 4.3.2 Sémantika zovšeobecnej substitúcie

Sémantika zovšeobecnenej substitúcie je definovaná v zmysle predikátových transformérov a kalkulu najslabšej pre-podmienky (Weakest precondition calculus) E.W. Dijkstru [8] .

Najslabšiu pre-podmienku zapisujeme $[S]P$, čo je ekvivalentné zápisu $wp(S, P)$ v Dijkstrovom kalkule. To znamená, že ak výpočet $S$ ($S$ je nejaká GS) započne v stave spĺňajúcom predikát $[S]P$, určite terminuje a to v stave (post-stave) spĺňajúcom predikát $P$.

Definícia predikátu $[S]P$ pre základnú substitúciu, jednoduché priradenie $x := e$, kde $x = (x_1, \ldots, x_n)$ je n-tica (zoznam) premenných a $e = (e_1, \ldots, e_n)$ je n-tica (zoznam) výrazov, rovnakej dĺžky ako $x$, je

$$[x := e]P \equiv P[e/x] \tag{4.1}$$

Predikát[6] $P[e/x]$ znamená textuálnu substitúciu každého z výrazov $e_1, \ldots, e_n$ za zodpovedajúci výraz $x_1, \ldots, x_n$. Ak sa v niektorom $e_i$ nachádza premenná, ktorá sa po vykonaní substitúcie stane viazanou vo výslednom predikáte, potom kvantifikovaná premenná, ktorá by viazala túto premennú je premenovaná, aby sa vyhlo premenným z $e_i$.

Definícia $[S]P$ pre ostatné GS je vytvorená štrukturálnou indukciou zo základnej substitúcie (tabuľka 4.12).

V prípade $[@v.S]P$ premenná $v$ nie je voľná v $P$ a v poslednom prípade $\gamma$ je nová premenná, ktorá nie je voľná v substitúcii $WHILE$ alebo v zahrnutých predikátoch a $l$ je zoznam premenných modifikovaných v cykle.

Význam týchto definícií zdá sa byť jasný, podrobnejšie vysvetlenie si žiada snáď len sémantika konštruktu WHILE, predstavujúceho štandardný cyklus. Význam jednotlivých častí jeho predikátu $[S]P$ je nasledovný:

1. Invariant cyklu $I$ platí pred začatím vykonávania cyklu. V praxi často máme, že premenné používané v cykle sú pred vstupom doň iniciované nejakou GS $T$ a teda potrebujeme vypočítať najslabšiu pre

   $$T; \text{WHILE } E \text{ DO } S \text{ INVARIANT } I \text{ VARIANT } ve \text{ END.}$$

   Potom namiesto $I$ počítame $[T]I$.

2. Invariant cyklu $I$ je zachovávaný substitúciou cyklu (telom cyklu) $S$ za predpokladu, že sa vstúpilo do cyklu. $I$ je vždy pravdivý na začiatku každého vykonávania tela cyklu a aj pri ukončení cyklu.

3. Invariant spolu s negáciou testu cyklu ($\neg E$) implikuje post-podmienku.

[6] Zápis $P[e/x]$ sa používa v [22]. V [1] sa predikát $P[e/x]$ zapisuje ako $P[x := e]$.

| | |
|---|---|
| $[x := e]P$ | $\equiv$ P[e/x] |
| $[SKIP]P$ | $\equiv P$ |
| $[S_1 [] S_2]P$ | $\equiv [S_1]P \wedge [S_2]P$ |
| $[E \mid S]P$ | $\equiv E \wedge [S]P$ |
| $[E \Longrightarrow S]P$ | $\equiv E \Rightarrow [S]P$ |
| $[@v.S]P$ | $\equiv \forall v : [S]P$ |
| $[S_1; S_2]P$ | $\equiv [S_1][S_2]P$ |
| [IF $E$ THEN $S_1$ ELSE $S_2$ END]$P$ | $\equiv (E \Rightarrow [S_1]P) \wedge (\neg E \Rightarrow [S_2]P)$ |
| [WHILE $E$ DO $S$ | |
| INVARIANT $I$ | |
| VARIANT $ve$ END]$P$ | $\Leftarrow$ |
| | 1. $I \wedge$ |
| | 2. $\forall l : (I \wedge E \Rightarrow [S]I) \wedge$ |
| | 3. $\forall l : (I \wedge \neg E \Rightarrow P) \wedge$ |
| | 4. $\forall l : (I \wedge E \Rightarrow ve \in N) \wedge$ |
| | 5. $\forall l : (I \wedge E \wedge (ve = \gamma) \Rightarrow [S](ve < \gamma))$ |

Tabuľka 4.12: Sémantika GS

4. Počas vykonávania tela cyklu, variant $ve$ je vždy prirodzené číslo. Obyčajne je výrazom zahŕňajúcim premenné modifikované v tele cyklu.

5. Variant $ve$ je vždy striktne zmenšený pri každom vykonaní tela cyklu.

Z bodov 3 a 4 vyplýva, že cyklus bude terminovať po konečnom počte iterácií.

Najslabšia pre-podmienka je, celkom prirodzene, kritériom pre porovnávanie zovšeobecnených substitúcií (definícia 4.3.1).

**Definícia 4.3.1.** *Nech $S_1$ a $S_2$ sú GS. Potom $S_1$ je rovná $S_2$ práve vtedy, ak pre každý predikát $P$ platí, že $[S_1]P$ je ekvivalentné $[S_2]P$:*

$$(S_1 = S_2) \Leftrightarrow \forall P.([S_1]P \equiv [S_2]P)$$

Na základe predchádzajúcej definície je možné sformulovať niekoľko *zákonov ekvivalencie pre GS*. Je ich možné nájsť v časti 6.1 publikácie [1] a niektoré z nich sú uvedené v tabuľke 4.13.

V tejto kapitole sme sa nezaoberali sémantikou viacnásobnej GS. Tá je definovaná odlišne od ostatných a pred jej uvedením je potrebné zaviesť niekoľko predikátov charakterizujúcich GS.

| | | |
|---|---|---|
| $P \Longrightarrow (Q \Longrightarrow S)$ | $=$ | $(P \wedge Q) \Longrightarrow S$ |
| $P \Longrightarrow (S [] T)$ | $=$ | $(P \Longrightarrow S) [] (P \Longrightarrow T)$ |
| $P \Longrightarrow (@v.S)$ | $=$ | $@v.(P \Longrightarrow S)$ |
| $@v.S [] @v.T$ | $=$ | $@v.(S [] T)$ |

Tabuľka 4.13: Niektoré zákony ekvivalencie pre GS

### 4.3.3 Predikáty charakterizujúce GS

Pre každú GS $S$ možno definovať niekoľko dôležitých predikátov, ktoré ju charakterizujú.

Prvým predikátom je $fis(S)$, ktorý sa nazýva *podmienka uskutočniteľnosti* (feasibility codition). Charakterizuje množinu stavov[7] z ktorých je vykonávanie $S$ prípustné, ale nemusí nevyhnutne terminovať, pretože post-stav v tomto prípade môže reprezentovať nedefinovaný stav, ktorý nespĺňa žiaden predikát. Predikát $fis(S)$ je definovaný takto:

$$fis(S) \equiv \neg[S]false \tag{4.2}$$

Druhý predikát je $trm(S)$, *podmienka ukončenia* (temination codition). Charakterizuje množinu stavov z ktorých ak začne vykonávanie (výpočet) $S$, tak určite terminuje. Je definovaný nasledovne:

$$trm(S) \equiv [S]true \tag{4.3}$$

Kvôli ilustrácii sú tvary predikátov $fis$ a $trm$ niektorých GS uvedené v tabuľke 4.14.

| **S** | **fis(S)** | **trm(S)** |
|---|---|---|
| $x := e$ | $true$ | $true$ |
| SKIP | $true$ | $true$ |
| $S_1 [] S_2$ | $fis(S_1) \vee fis(S_2)$ | $trm(S_1) \wedge trm(S_2)$ |
| $P \mid S_1$ | $P \Rightarrow fis(S_1)$ | $P \wedge trm(S_1)$ |
| $P \Longrightarrow S_1$ | $P \wedge fis(S_1)$ | $P \Rightarrow trm(S_1)$ |
| $@v.S_1$ | $\exists v : fis(S_1)$ | $\forall v : trm(S_1)$ |
| $@v.(P \Longrightarrow S_1)$ | $\exists v : (P \wedge fis(S_1))$ | $\forall v : (P \Rightarrow trm(S_1))$ |

Tabuľka 4.14: Predikáty $fis$ a $trm$ niektorých GS

[7]Hovoríme, že predikát $P$ charakterizuje stav $s$, ak $P$ v stave $s$ platí.

Tretím predikátom, mierne odlišnej povahy ako predchádzajúce dva, je $prd_x(S)$, Nazýva sa nazýva *pred-po predikát* (before-after predicate) a definuje ako vykonanie $S$ zmení stav špecifikačného komponentu, v ktorom sa vyskytuje. Je definovaný rovnicou (4.4).

$$prd_x(S) \equiv \neg[S](x \neq x') \tag{4.4}$$

Symbol $x$ v (4.4) predstavuje zoznam stavových premenných špecifikačného komponentu v ktorom sa $S$ vyskytuje a $x'$ zoznam premenných, iných ako stavových, rovnakej dĺžky ako $x$. Každá premenná z $x'$ má rovnaký typ, ako príslušná premenná z $x$ a jej meno je zvyčajne menom príslušnej premennej z $x$ doplneným apostrof. V $prd_x(S)$ vlastne $x$ reprezentuje stav špecifikačného komponentu pred vykonaním $S$ a $x'$ stav po vykonaní $S$. Tvar predikátu $prd$ pre niektoré GS možno nájsť v tabuľke 4.15.

| **S** | $\mathbf{prd_x(S)}$ |
|---|---|
| $x := e$ | $x' = e$ |
| SKIP | $x' = x$ |
| $S_1 [] S_2$ | $prd_x(S_1) \vee prd_x(S_2)$ |
| $P \mid S_1$ | $P \Rightarrow prd_x(S_1)$ |
| $P \Longrightarrow S_1$ | $P \wedge prd_x(S_1)$ |
| $@v.S_1$ | $\exists v : prd_x(S_1)$, $v$ nie je voľné v $x'$ |
| $@v.(P \Longrightarrow S_1)$ | $\exists v : (P \wedge prd_x(S_1))$, $v$ nie je voľné v $x'$ |

Tabuľka 4.15: Predikát $prd_x$ niektorých GS

Na prvý pohľad sa môže zdať, že dvojitá negácia, ktorú predikát prd obsahuje je zbytočná a že $\neg[S](x \neq x') \equiv [S](x = x')$. To však platí iba v prípade GS, ktoré sú deterministické, vždy vykonateľné a vždy terminujúce (tabuľka 4.16). Pre ostatné GS sú $\neg[S](x \neq x')$ a $[S](x = x')$ rozdielne, príklady takýchto GS sú uvedené v tabuľke 4.17.

| **S** | $\neg[\mathbf{S}](\mathbf{x} \neq \mathbf{x}')$, $[\mathbf{S}](\mathbf{x} = \mathbf{x}')$ |
|---|---|
| $x := 3$ | $x' = 3$ |
| $(x > 0 \Longrightarrow x := 5)[]$<br>$(\neg(x > 0) \Longrightarrow x := 3)$ | $(x > 0 \wedge x' = 5) \vee$<br>$(\neg(x > 0) \wedge x' = 3)$ |

Tabuľka 4.16: Príklady GS s rovnakými $prd_x$ a $[S](x = x')$

| **S** | $\neg[\mathbf{S}](\mathbf{x} \neq \mathbf{x}')$ | $[\mathbf{S}](\mathbf{x} = \mathbf{x}')$ | **Dôvod rozdielu** |
|---|---|---|---|
| $x := 3 [] x := 5$ | $x' = 3 \vee x' = 5$ | $x' = 3 \wedge x' = 5$ | nedeterminizmus $S$ |
| $x > 0 \mid x := 5$ | $x > 0 \Rightarrow x' = 5$ | $x > 0 \wedge x' = 5$ | $trm(S) \not\equiv true$ |
| $x > 0 \Longrightarrow x := 5$ | $x > 0 \wedge x' = 5$ | $x > 0 \Rightarrow x' = 5$ | $fis(S) \not\equiv true$ |
| $@v.(v > 0$ $\Longrightarrow x := v)$ | $\exists v.(v > 0$ $\wedge x' = v)$ | $\forall v.(v > 0$ $\Rightarrow x' = v)$ | nedeterminizmus $S$ a $fis(S) \not\equiv true$ |

Tabuľka 4.17: Príklady GS s rôznymi $prd_x$ a $[S](x = x')$

### 4.3.4 Sémantika viacnásobnej GS

Sémantika viacnásobnej GS, $S_1 \| S_2$, je pomocou najslabšej pre-podmienky definovaná iba pre prípad

$$x_1 := e_1 \| x_2 := e_2 \tag{4.5}$$

keďže

$$x_1 := e_1 \| x_2 := e_2 \quad = \quad x_1, x_2 := e_1, e_2 \tag{4.6}$$

Najslabšiu pre-podmienku pre GS (4.5) teda vypočítame podľa vzťahu (4.1), kde $x = (x_1, x_2)$ a $e = (e_1, e_2)$. GS (4.5) nazývame *viacnásobná základná substitúcia*.

V ostatných prípadoch je potrebné upraviť danú GS na tvar (4.6) použitím vlastností viacnásobnej GS (tabuľka 4.18). Vlastnosť (d) platí len za predpokladu, že $trm(S) = true$ a v (e) $v$ nie je voľné v $S$. Obmedzenie použiteľnosti vlastnosti (e) možno jednoducho odstrániť premenovaním premenných zo zoznamu $v$.

| | | | |
|---|---|---|---|
| (a) | $S \| \text{SKIP}$ | $=$ | $S$ |
| (b) | $S \| (P \mid T)$ | $=$ | $P \mid (S \| T)$ |
| (c) | $S \| (T [] U)$ | $=$ | $S \| T [] S \| U$ |
| (d) | $S \| (P \Longrightarrow T)$ | $=$ | $P \Longrightarrow S \| T$ |
| (e) | $S \| @v.T$ | $=$ | $@v.(S \| T)$ |

Tabuľka 4.18: Vlastnosti viacnásobnej GS

Viacnásobnú GS, $S_1 \| S_2$, je možné tiež úplne definovať použitím predikátov $trm$ a $prd$ [1]:

$$trm(S_1 \| S_2) \equiv trm(S_1) \wedge trm(S_2) \tag{4.7}$$

$$prd_{x,y}(S_1 \| S_2) \equiv prd_x(S_1) \wedge prd_y(S_2) \tag{4.8}$$

Symbol $x$ v (4.8) predstavuje zoznam premenných špecifikačného komponentu, v ktorom sa vyskytuje GS $S_1$ a $y$ zoznam premenných komponentu, v ktorom sa vyskytuje $S_2$.

Zo vzťahov (4.7) (4.8) je možné odvodiť nasledujúci závažný výsledok o povahe viacnásobnej GS [1]:

**Veta 4.3.1.** *Nech $S$, $T$ sú GS, pracujúce s rôznymi premennými $x$ a $y$, a $P$, $Q$ sú predikáty také, že $x$ nie je voľné v $P$ a $y$ nie je voľné v $Q$ (tzn. $x \backslash P$ a $y \backslash Q$). Potom*

$$[S]P \wedge [T]Q \Rightarrow [S \| T](P \wedge Q)$$

### 4.3.5 Normálna forma GS

Podľa [1] je možné každú GS, ktorá sa vyskytuje v nejakom abstraktnom stroji zapísať v normánej forme uvedenej vo vete 4.3.2.

**Veta 4.3.2.** *Nech $S$ je zovšeobecnená substitúcia, $x$ je zoznam stavových premenných abstraktného stroja, v ktorom sa $S$ vyskytuje, $x'$ zoznam premenných, iných ako stavových, rovnakej dĺžky ako $x$ a $P$, $Q$ sú predikáty, $x'$ nie je voľné v $P$. Potom*

$$S = P | @x'.(Q \Longrightarrow x := x')$$

*pre nejaké $P$ a $Q$, kde $x'$ nie je voľné v $P$.*

Dôkaz vety 4.3.2 pre GS ":=", SKIP, PRE, CHOICE, SELECT a ANY je uvedený v [1].

Taktiež v [22] sa uvázda, že každá GS môže byť vyjadrená v normálnej forme v tvare:

```
PRE P
THEN
    ANY w
    WHERE Q
    THEN
        S1
    END
end
```

Táto normálna forma korešponduje s vetou 4.3.2 (jediným rozdielom je, že GS $S_1$ nie je bližšie špecifikovaná) a jej dodržiavanie je dôležité pre správne automatické generovanie povinných dôkazov v nástrojoch pre B-Metódu.

Použitím predikátov z časti 4.3.3 môžeme bližšie určiť normálnu formu z vety 4.3.2:

**Veta 4.3.3.** *Nech $S$, $x$ a $x'$ sú rovnaké ako v prípade vety 4.3.2. Potom*

$$S = trm(S)|@x'.(prd_x(S) \Longrightarrow x := x')$$

Dôkaz vety 4.3.3 je tiež uvedený v [1]. Ako dôsledok vety môžeme konštatovať [1], že predikáty $trm(S)$ a $prd_x(S)$ kompletne charakterizujú GS $S$. Takisto dobre môžeme povedať, že $S$ charakterizujú predikáty $trm(S)$ a $trm(S) \Rightarrow prd_x(S)$.

## 4.4 Abstraktný stroj - Machine

Ako už bolo spomenuté vyššie, formálna špecifikácia systému je v B-AMN popísaná kolekciou abstraktných strojov, prepojených kompozičnými mechanizmami (kapitola 4.7), vďaka ktorým možno popísať aj značne rozsiahle systémy.

*Abstraktný stroj* (MACHINE), skrátene *B-stroj*, je svojou koncepciou blízky triede v objektovo orientovanom programovaní, či skôr balíku (package) v jazyku ADA. Slúži na zjednotenie (zapuzdrenie) kolekcie matematických prvkov, konštánt, množín, stavových premenných a súboru operácií na týchto premenných do pomenovaného modulu, ktorý potom podľa potreby môže byť videný inými komponentmi, alebo do nich začlenený. Premenné stroja môžu byť modifikované iba operáciami tohto stroja a nie operáciami iných strojov (okrem prípadu ak tieto operácie volajú operácie pôvodného stroja). Účelom tohto obmedzenia je najmä zjednodušenie povinných dôkazov stroja. Navyše to zapadá do konceptu B-stroja ako komponentu "vlastniaceho" lokálne údaje a poskytujúceho operácie potrebné na manipuláciu s nimi a prístup k nim. Vo všeobecnosti má zápis abstraktného stroja (bez kompozičných klauzúl SEES,USES,...) formu:

MACHINE M($p$)
CONSTRAINTS $C$
SETS $St$
CONSTANTS $k$
PROPERTIES $B$
VARIABLES $v$
DEFINITIONS $D$
INVARIANT $I$
ASSERTIONS $A$
INITIALISATION $T$
OPERATIONS
  $y \leftarrow op(x) \mathrel{\widehat{=}}$

```
        PRE P
        THEN S
    END
    ...
END
```

### 4.4.1 Syntax a sémantika klauzúl stroja

Uvedené klauzuly majú nasledujúcu syntax a sémantiku:

- *Parametre (parameters - p)*: Stroj M môže byť parametrizovaný zoznamom parametrov *p*. Parametre sú *skalárne* alebo *množinové*. Hodnotou skalárneho parametra môže byť napríklad prirodzené číslo, alebo prvok z množiny definovanej v klauzule SETS, či z množiny, ktorá je množinovým parametrom. Skalárne parametre sa používajú napríklad na obmedzenie rozsahu hodnôt premenných stroja. Množinový parameter je množina a predstavuje nový typ v danom abstraktnom stroji. Meno množinového parametra nesmie obsahovať malé písmená.

- *Obmedzenia (*CONSTRAINTS*)* je klauzula definujúca logické vlastnosti parametrov *p*. Jej súčasťou musí byť pre každý skalárny parameter predikát určujúci jeho typ[8]. Obmedzenia stroja by nemali spájať rôzne množinové parametre, pretože tieto parametre sú považované za navzájom nezávislé.

- *Množiny (*SETS*)*: Klauzula SETS má tvar:

  $\mathsf{setdef}_1$
  ...
  $\mathsf{setdef}_n$

  Každá definícia $\mathsf{setdef}_i$ určuje novú množinu ako *enumeračnú* (enumerated set), alebo *odloženú* (deferred set). Enumeračná množina je určená vymenovaním jej prvkov a jej definícia (ak $SS$ je meno množiny a počet jej prvkov je $n$) má tvar:

  $$SS = \{val_1, \ldots val_n\}$$

  Definícia odloženej množiny obsahuje iba meno množiny, bez bližšej informácie o jej prvkoch. O odloženej množine predpokladáme, že je konečná a neprázdna.

[8]Typ výrazu v B-AMN je najväčšia množina, ktorej je daný výraz prvkom.

- *Konštanty (*CONSTANTS*)*: Táto klauzula obsahuje zoznam identifikátorov $con_1, \ldots, con_n$, ktoré definujú $n$ dátových prvkov, ktoré sú poskytované operáciám stroja v režime len na čítanie (read-only).

- *Vlastnosti (*PROPERTIES*)*: Tu sú určené logické vlastnosti množín a konštánt. Obsahuje predikát, obyčajne konjunkciu formúl, zahŕňajúcu iba zavedené konštanty a množiny. Typové obmedzenie $c \in T$ ($T$ je množina) alebo $c = value$ musí byť prítomné pre každú konštantu $c$, ktorá sa vyskytuje v klauzule CONSTANTS.

- *Stavové premenné (*VARIABLES*)*: Sú definované zoznamom $var_1, \ldots, var_n$ identifikátorov stavových premenných stroja. Stavové premenné sú lokálnymi údajmi stroja, vyjadrujú jeho stav, a ich hodnota nemôže byť priamo menená operáciami iného stroja (ich obsah možno "z vonku" zmeniť len volaním príslušnej operácie toho B-stroja, v ktorom sú definované).

- *Invariant (*INVARIANT*)*: Určuje obmedzenia na premenné, vrátane definovania ich typov[9]. Pozostáva z predikátu, zvyčajne konjunkcie predikátov, zahŕňajúcich identifikátory premenných, množín, konštánt a parametrov stroja. Invariant stroja $M$ neobsahuje invarianty strojov, ktoré sú prístupné stroju $M$.

- *Tvrdenia (*ASSERTIONS*)*: Táto klauzula sa používa na deklaráciu dodatočných invariantných vlastností $A$ stroja a tiež na vyjadrenie validačných vlastností.

- *Definície (*DEFINITIONS*)*: Klauzula obsahuje zoznam definícií matematických skratiek v tvare:

  $\mathsf{abbdef}_1$
  ...
  $\mathsf{abbdef}_n$

  Každá definícia $\mathsf{abbdef}_i$ má tvar $data_i == expr_i$, kde $data_i$ je identifikátor alebo meno parametrizovanej funkcie $f(parametre)$ a $expr_i$ je B-AMN výraz, predikát alebo GS a môže obsahovať viditeľné premenné, konštanty a parametre $f$. Tieto definície slúžia ako makro-definície, napríklad

  $\mathsf{sqr(p)} == \mathsf{((p)*(p))}$

[9]Invarianty, ktoré len definujú typy premenných nazývame *elementárne*.

definuje funkciu druhej mocniny parametra $p$. Operátor "==" znamená "prepísať na".

- *Inicializácia (*INITIALISATION*)*: Je to zovšeobecnená substitúcia, ktorej vykonaním sa každej stavovej premennej priradí počiatočná hodnota, čím sa B-stroj dostane do počiatočného stavu. Inicializácia môže byť aj nedeterministická.

- *Operácie (*OPERATIONS*)*: Obsahuje zoznam definícií operácií v tvare:

  $\mathsf{opheader}_1 \cong \mathsf{opdef}_1$
  ...
  $\mathsf{opheader}_n \cong \mathsf{opdef}_n$

  Hlavička operácie, opheader, má vo všeobecnosti formu: $y \leftarrow opname(x)$. Vstupné parametre $(x)$ a výstupné parametre $(y)$ sú voliteľne a môžu chýbať. Opdef je zovšeobecnená substitúcia, ktorá má vo všeobecnosti tvar

  PRE $P$ THEN $S$ END

  kde $P$ je predikát definujúci pre-podmienku operácie a $S$ je GS tvoriaca telo operácie. Ak $P = true$, zvykne sa opdef zapisovať iba ako $S$, alebo ako

  BEGIN $S$ END .

V rámci inicializácie a operácií abstraktného stroja môžu byť použité všetky zovšeobecnené substitúcie okrem sekvenčnej kompozície (;) a konštruktov cyklu WHILE. Dôvodom pre ich vylúčenie je požiadavka podrobne definovať operácie tak, aby úzko súviseli so stavovými prechodmi, ktoré majú byť dosiahnuté. Abstraktná špecifikácia by mala byť jasne zrozumiteľná a nemala by vyžadovať analýzu medzistavov, ktoré môžu vzniknúť použitím ; alebo WHILE. Ich vylúčenie tiež zdôrazňuje deklaratívnu povahu MACHINE: operácie abstraktného stroja majú určovať *čo* sa má urobiť a nie *ako* sa to má urobiť. V MACHINE nemožno používať ani konštrukt VAR.

### 4.4.2 Povinné dôkazy

Aby sme dokázali, že špecifikácia abstraktného stroja je vnútorne konzistentná, musíme vykonať *povinné dôkazy* (POb) uvedené v tabuľke 4.19. Prvé tri dôkazy preverujú, či vôbec existujú údajové členy vyhovujúce daným podmienkam. Dôkazy (4) a (5) overujú nastolenie invariantu pri inicializácii a jeho zachovania operáciami stroja. Konkrétnejšie:

| | |
|---|---|
| (1) | $\exists p.C$ |
| (2) | $C \Rightarrow \exists(St, k).B$ |
| (3) | $B \wedge C \Rightarrow \exists v.I$ |
| (4) | $B \wedge C \Rightarrow [T]I$ |
| (5) | $B \wedge C \wedge I \wedge P \Rightarrow [S]I$ |

Tabuľka 4.19: Povinné dôkazy abstraktného stroja

(1) *existencia parametra* - preveruje, či existujú hodnoty parametra stroja $p$ spĺňajúce obmedzenia $C$. Ak nie, potom tento stroj nemôže byť správne inštanciovaný a neexistuje ani program vyhovujúci danej abstraktnej špecifikácii.

(2) *existencia konštánt a množín* - preveruje, či existujú konštanty $k$ a množiny $St$ vyhovujúce vlastnostiam $B$. V opačnom prípade neexistuje program vyhovujúci špecifikácii.

(3) *existencia stavu stroja* - overuje, či vzhľadom na vlastnosti $B$ a obmedzenia $C$ stroja, existujú nejaké hodnoty stavových premenných $v$, ktoré spĺňajú invariant $I$.

(4) *inicializácia* - overuje, či inicializácia $T$ zavádza platnosť invariantu $I$, vzhľadom na vlastnosti $B$ a obmedzenia $C$ stroja. Ak $T$ túto podmienku nespĺňa, potom je narušený základný predpoklad platnosti invariantu $I$ počas životnosti implementácie stroja.

(5) *zachovanie invariantu* - overuje, či každá operácia (ktorej telo tvorí zovšeobecnená substitúcia $S$) zachováva platnosť invariantu $I$, ak je volaná za platnosti svojej pre-podmienky $P$. Ak to neplatí, potom nie je záruka, že invariant $I$ je pravdivý počas celej životnosti implementácie stroja.

Dôkaz (5) je potrebné vykonať pre každú operáciu stroja.

| | |
|---|---|
| (5) | $B \wedge C \wedge I \wedge P \wedge A \Rightarrow [S]I$ |
| (6) | $B \wedge C \wedge I \Rightarrow A$ |

Tabuľka 4.20: Modifikované POb pre stroj s ASSERTIONS

Ak je použitá klauzula ASSERTIONS, modifikuje sa piaty a pridáva šiesty dôkaz (tabuľka 4.20).

### 4.4.3 Abstraktné a konkrétne údaje

To, čo sme o B-stroji povedali doteraz odráža stav B-AMN používanej v nástroji *B-Toolkit* [29, 22]. V novšej verzii B-AMN pre *Atelier B* [30, 1] nájdeme menšie rozdiely, týkajúce sa najmä konštánt a premenných. Presnejšie, v *Atelier B* rozlišujeme *abstraktné* a *konkrétne* konštanty a premenné.

*Konkrétne konštanty a premenné* musia byť priamo implementovateľné. Preto ich typom môže byť iba

$$S, \quad S \rightarrow T, \quad A_1 \times \ldots \times A_n \rightarrow T, \quad \mathbb{P}(S)$$

kde $S, T, A_1 \ldots A_n$ sú skalárne množiny (tzn. množina prirodzených prípadne celých čísel, množinové parametre či množiny definované v klauzule SETS). Na *abstraktné konštanty a premenné* žiadne takéto obmedzenia kladené nie sú.

Klauzula uvádzajúce konkrétne konštanty sa označuje CONCRETE_CONSTANTS alebo iba CONSTANTS a klauzula uvádzajúce abstraktné je ABSTRACT_CONSTANTS.

V prípade stavových premenných máme ABSTRACT_VARIABLES alebo iba VARIABLES pre abstraktné a CONCRETE_VARIABLES pre konkrétne.

## 4.5 Zjemnenie - Refinement

Zjemnenie je procesom prechodu od abstraktnej špecifikácie stroja k menej abstraktnej špecifikácii, pomocou transformácie jej operácií alebo údajov. Zjemňovaním vlastne postupne "prerábame" abstraktnú špecifikáciu do implementovateľnej podoby, pričom je nutné, aby vonkajšie správanie zjemnenej komponenty ostalo rovnaké ako, resp. nerozoznateľné od, správania zjemňovanej komponenty. V rámci zjemňovania tiež môžeme do špecifikácie zahrnúť viac detailov z pôvodného, neformálneho, opisu systému. Krok zjemnenia špecifikácie môže byť od abstraktného stroja k zjemneniu, od zjemnenia k zjemneniu, alebo od zjemnenia k implementácii (obrázok 4.1 b).
Špecifikácia zjemnenia $N$ sa líši od špecifikácie stroja $M$ v tom že:

- musí obsahovať klauzulu REFINES M, čím označuje *jedinú* komponentu $M$, ktorú zjemnenie zjemňuje,
- musí mať identickú množinu hlavičiek operácií ako komponent, ktorý zjemňuje,
- nemá explicitné parametre[10] (tie sú dané zjemňovaným komponentom),

[10] V B-AMN "*francúzkej školy B*" (Atelier B) [30, 1] sa v zjemnení uvádza aj zoznam parametrov, ktorý musí byť rovnaký ako v zjemňovanom komponente. Klauzula CONSTRAINTS sa však taktiež vynecháva.

- v operáciách a inicializácii možno použiť všetky zovšeobecnené substitúcie použiteľné v abstraktnom stroji a tiež sekvenčnú kompozíciu (;),
- niektorý z konjunktov invariantu zjemnenia identifikuje, ako stav zjemnenia vyjadruje stav zjemňovaného komponentu.
- v operáciách zjemnenia je možné volať opytovacie operácie[11] videného stroja.

Platí, že iba jediný komponent môže byť zjemnený daných konkrétnym zjemnením a v klauzule REFINES zjemnenia (alebo implementácie) $N$ stroja $M$ nie sú povolené parametre. Je to preto, že zjemnenie a zjemňovaný komponent považujeme za dva alternatívne popisy toho istého systému, ktoré majú rovnaké vonkajšie správanie (rozhranie) z hľadiska parametrov, množiny operácií a mien operácií. Vnútorná implementácia operácií je však rozdielna.

V procese zjemnenia môže byť zmenený kód inicializácie a operácií (*procedurálne zjemnenie*) a tiež údajové členy komponentu. *Procedurálne zjemnenie* je formálne definované pomocou operátora "$\sqsubseteq$" nasledovne:

**Veta 4.5.1.** *Nech $S_1$ a $S_2$ sú GS a $P$ predikát. Potom $S_2$ zjemňuje $S_1$ (zapisujeme $S_1 \sqsubseteq S_2$) práve vtedy, ak*

$$\forall P.[S_1]P \Rightarrow [S_2]P$$

Typickým procedurálnym zjemnením je oslabenie pre-podmienky či obmedzenie nedeterminizmu.

Zápis zjemnenia $N$, ktoré zjemňuje abstraktný stroj $M(p)$ (kapitola 4.4) má v jazyku B-AMN formu:

REFINEMENT N
REFINES M
SETS $St1$
CONSTANTS $k1$
PROPERTIES $B1$
VARIABLES $w$
DEFINITIONS $D1$
INVARIANT $J$
ASSERTIONS $A1$
INITIALISATION $T1$
OPERATIONS

[11] Opytovacie operácie (enquiry operations) sú operácie, ktoré vracajú hodnoty premenných stroja, ale nemenia ich.

$y \leftarrow op(x) \mathrel{\widehat{=}}$
PRE $P1$
THEN $S1$
END
...
END

Predikát $J$ v invariante zjemnenia obsahuje určenie typov a obmedzení lokálnych premenných $w$ a *reláciu zjemnenia* medzi premennými $w$ a $v$. *Relácia zjemnenia*, $R$, vyjadruje vzťah medzi stavmi zjemňovaného komponentu a stavmi jeho zjemnenia.

### 4.5.1 Povinné dôkazy

Povinné dôkazy zjemnenia sú uvedené v tabuľke 4.21. Význam dôkazov je

| | |
|---|---|
| (1) | $C \wedge B \wedge B1 \Rightarrow \exists(v, w).(I \wedge J)$ |
| (2) | $C \wedge B \wedge B1 \Rightarrow [T1]\neg[T]\neg J$ |
| (3) | $C \wedge B \wedge B1 \wedge I \wedge J \wedge P \Rightarrow P1 \wedge [S1']\neg[S]\neg(J \wedge y' = y)$ |

Tabuľka 4.21: Povinné dôkazy zjemnenia

nasledujúci:

(1) *existencia zmiešaného stavu* - existuje kombinovaný abstraktný a konkrétny stav, ktorý vyhovuje relácii zjemnenia a invariantu abstraktného stroja. V prípade neplatnosti nebude existovať vykonateľná implementácia abstraktnej špecifikácie prostredníctvom tohto zjemnenia.

(2) *zjemnenie inicializácie* - ak platí, potom inicializácia zjemnenia $T1$ ustanoví situáciu, v ktorej inicializácia špecifikácie stroja $T$ nemôže zlyhať pri stanovení podmienok $J$.

(3) *zjemnenie operácie* - zabezpečenie korektnosti operácií a verifikácia, či operácia $S1$ zjemnenia stanoví situáciu, v ktorej operácia $S$ zjemňovaného komponentu nemôže zlyhať pri splnení podmienok invariantu $J$. $S1'$ je $S1$, v ktorom výstupné premenné $y$ sú nahradené[12] premennými $y'$.

[12]Nahradenie je potrebné, aby boli jednoznačne odlíšené výstupné premenné operácie zjemneného ($y'$) a zjemňovaného komponentu ($y$).

Dôkaz (3) je potrebné vykonať pre každú operáciu zjemnenia.

Jedným z dôsledkov uvedených povinných dôkazov je, že zjemňovať možno iba plne uskutočniteľnú (feasible) operáciu. GS totiž možno zjemniť iba na tak isto, alebo menej uskutočniteľnú GS a operácie implementácie musia byť plne uskutočniteľné. Preto je dobré sa ešte pred zjemňovaním presvedčiť, či sú plne uskutočniteľné všetky operácie B-stroja. Formálny dôkaz uskutočniteľnosti operácií (tzn. či ich $fis \equiv true$) sa medzi POb abstraktného stroja nenachádza iba kvôli jeho vysokej náročnosti. Tá môže byť rovná vypracovaniu implementácie danej operácie.

## 4.6 Implementácia - Implementation

Implementácia stroja reprezentuje v B-Metóde posledný krok vývoja systému a je prerekvizitou, na základe ktorej je možné priamo generovať vykonateľný kód. Implementácie stroja určené na preklad zvyčajne importujú abstraktnú špecifikáciu existujúceho (vopred kódovaného) systému a to pomocou klauzuly IMPORTS, popísanej v časti 4.7. Klauzula PROPERTIES v implementácii stroja zabezpečuje[13], aby sa všetkým konštantám zjemňovaného komponentu priradila presná hodnota, vyhovujúca ich definíciám. Z tohto dôvodu môžu byť v implementácii stroja použité iba konkrétne konštanty (časť 4.4.3). Vstupom a výstupom operácie implementácie stroja môžu byť taktiež iba parametre implementovateľných typov.

Importované stroje (musia existovať ich korektne vykonateľné implementácie) musia byť inštanciované konkrétnymi množinami a skalárnymi parametrami pri zavedení do implementácie stroja. Tieto parametre často súvisia s parametrami implementovaného stroja. Ak je potrebné viacnásobné importovanie toho istého stroja, použije sa premenovanie na odlíšenie jeho kópií. Zo špecifikácie implementácie stroja boli pôvodne vynechané dve klauzuly:

- VARIABLES, pretože nie je možné automaticky generovať vykonateľný kód pre všeobecný prípad B-AMN premenných (napr. pre operácie na množinách a sekvenciách). Všetky premenné potrebné pri implementácii stroja sa získavajú importovaním (knižničných) strojov.

- INITIALISATION, pretože inicializácia implementácie je sekvenčnou kombináciou inicializácií importovaných strojov. Táto kombinácia musí zjemňovať inicializáciu komponentu, ktorý je zjemnený uvažovanou implementáciou. V prípade nevyhnutnosti môže byť klauzula INITIALISATION použitá na poskytnutie špecifickejšej inicializácie premenných importovaných strojov.

[13] V B-AMN pre Atelier B sa na tento účel používa špeciálna klauzula VALUES.

Toto obmedzenie však už celkom neplatí. V B-Metóde "*francúzkej školy B*" (Atelier-B), je možné použiť konkrétne premenné (časť 4.4.3). Tieto sú následne inicializované v klauzule INITIALISATION.

*Povinné dôkazy* implementácie sú zhodné s dôkazmi pre zjemnenie s tým rozdielom, že premenné importovaného stroja prevezmú miesto lokálnych premenných zjemnenia vzhľadom na zjemňovaný komponent.

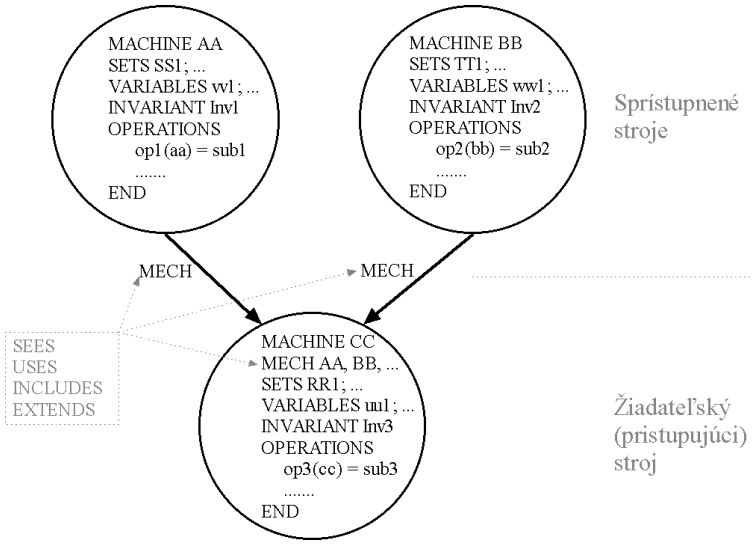
V operáciách implementácie stroja sú povolené nasledujúce formy GS:

- $v := e$, kde $v$ sú lokálne alebo výstupné premenné operácie;
- WHILE $E$ DO $S$ INVARIANT $I$ VARIANT $V$ END;
- IF $E$ THEN $S_1$ ELSE $S_2$ END a iné podmienkové príkazy, ktoré sú variantmi alebo komplexnejšími kombináciami tohto príkazu;
- VAR $v$ IN $S$ END. Lokálne premenné $v$ musia byť inicializované pred prvým čítaním;
- volania operácií importovaných strojov a opytovacích operácií videných strojov;
- sekvenčná kompozícia (;).

Týchto šesť zovšeobecnených substitúcií je postačujúcich na implementáciu ľubovoľného algoritmu. V B-AMN francúzkej školy sa môže v implementácii vyskytnúť aj GS pre-podmienka ale iba v podobe stráženého priradenia a stráženého volania operácie (tzn. PRE $P$ THEN $S$ END, kde $S$ je priradenie alebo volanie operácie).

## 4.7 Kompozičné mechanizmy

B-Metóda poskytuje 6 mechanizmov na vzájomné prepojenie (sprístupnenie) abstraktných strojov (M), zjemnení (R) a implementácií (I). V závislosti na spôsobe sprístupnenia, tzn. na zvolenom mechanizme, sa rôzne časti (operácie, premenné,...) sprístupňovaného (accessed) stroja stávajú na rôznych úrovniach "viditeľné" v žiadateľskom (accessing) stroji, alebo je s nimi nakladané, akoby boli priamo definované v žiadateľskom (accessing) stroji. Všeobecná schéma sprístupnenia prostriedkov jedného stroja iným strojom je znázornený na obrázku 4.2 ("MECH" je názov použitého mechanizmu). Na reprezentáciu sprístupnenia jedného stroja inému stroju sa v grafickom




Obr. 4.2: Všeobecná schéma sprístupnenia abstraktných strojov

zobrazení používajú hrany, označené názvom daného mechanizmu, orientované od sprístupneného k žiadateľskému stroju. Do špecifikácie žiadateľského stroja pribudne klauzula (klauzuly - jedna pre každý použitý mechanizmus) v tvare

MECH $\mathsf{m}_1, \ldots, \mathsf{m}_k$

$MECH$ je názov mechanizmu a $m_1, \ldots, m_k$ zoznam strojov týmto mechanizmom sprístupnených. K dispozícii sú mechanizmy SEES, USES, INCLUDES, EXTENDS, REFINES a IMPORTS:

- SEES je povolený v $M$, $R$ aj $I$[14], no videný môže byť iba $M$. V $M$ sprístupňuje množiny, konštanty a premenné tak, ako to ukazuje tabuľka 4.22. V $R$ a $I$ je možné použiť aj opytovacie operácie videného stroja. V $I$ však možno použiť (abstraktné) premenné videného stroja iba v invariantoch cyklov. Viac ako jeden $M$, $R$, $I$ môže vidieť daný abstraktný stroj. Hlavným účelom mechanizmu je podporiť oddelený vývoj subsystému, ktorý je (pomocou SEES) zdieľaný v read-only režime inými subsystémami v rámci aplikácie. Ak je stroj videný viacerými implementáciami, potom bude zvyčajne importovaný práve jednou.

- USES je povolený len v $M$. Jeden stroj môže byť použitý viacerými strojmi. Hlavným účelom USES je v sprostredkovaní zdieľaného prístupu

[14] $M$ - abstraktný stroj, $R$ - zjemnenie, $I$ - implementácia

k stavovým údajom ako SEES, z dôvodu uľahčenia konštrukcie špecifikácie vývoja subsystému. Ak stroj $A$ je používaný viacerými komponentmi špecifikácie, ktoré budú obsiahnuté (included) alebo rozšírené (extended) jednoduchou špecifikáciou stroja, potom stroj $A$ bude zvyčajne obsiahnutý v tomto stroji.
USES predstavuje silnejšiu formu prístupu ako SEES - premenné použitého stroja sa môžu v čítacom režime vyskytnúť v invariante používajúceho stroja, čo neplatí pre SEES. Táto silnejšia forma sprístupnenia zabraňuje použitým a používajúcim strojom, aby boli oddelene zjemnené až do vykonateľného kódu. Podobne ako u SEES ani v prípade USES nemôže pristupujúci stroj použiť aktualizačné operácie (update operations) sprístupneného a meniť tak jeho stav (hodnoty jeho premenných).
Stav použitého stroja môže byť zmenený operáciami subsystému v ktorom sa vyskytuje, pretože hlavný stroj, ktorý špecifikuje subsystém zvyčajne obsiahne alebo rozšíri tento stroj a tým získa prístup k jeho operáciám.

- INCLUDES (obsiahnutie) a EXTENDS (rozšírenie) je možné použiť iba v $M$. Ak stroj $M_1$ vložíme do $M_2$ ($M_2$ INCLUDES $M_1$), potom vlastne získame nový stroj, ktorého klauzuly SETS až INITIALISATION vzniknú zreťazením či konjunkciou obsahov príslušným klauzúl $M_1$ a $M_2$.
  Operácie $M_1$ je možné volať z operácií $M_2$, avšak maximálne 1 operáciu $M_1$ z danej operácie $M_2$. Volanie operácie má rovnakú syntax ako hlavička operácie. Samozrejme, formálne vstupné a výstupné parametre operácie sú nahradené skutočnými. Niektoré operácie $M_1$ sa môžu stať operáciami $M_2$ (tzn. môžu byť zavedené do $M_2$) - je ich však potrebné uviesť v klauzule PROMOTES v $M_2$. Pri vložení sú formálne parametre $M_1$ inštanciované (nahradené) aktuálnymi. Ak je potrebné viacnásobné vloženie toho istého stroja, použije sa premenovanie na odlíšenie jeho kópií.
  Klauzula EXTENDS znamená INCLUDES so zavedením všetkých operácií sprístupneného stroja.

  Iba jeden stroj môže obsiahnuť alebo rozšíriť daný stroj. Hlavným prínosom použitia INCLUDES je podpora hierarchického rozvrstvenia subsystémov (vývoja subsystému), zatiaľ čo EXTENDS korešponduje s dedením v objektovo orientovanom programovaní. Skupina strojov obsiahnutá alebo rozšírená daným strojom sa označuje pojmom *siblings*.

- REFINES sa používa v $R$ a $I$ a identifikuje komponent (M alebo $R$), ktorý je komponentom, v ktorom sa táto klauzula nachádza, zjemnený.

Stroj alebo zjemnenie môže byť zjemnené dvoma alebo viacerými spôsobmi, ale zjemnenie a implementácia môžu byť zjemnením práve jedného komponentu.

- IMPORTS sa používa iba v $I$. Importovať je možné iba abstraktné stroje. Voliteľná klauzula PROMOTES označuje zavedené operácie z importovaného stroja do importujúcej implementácie. Práve jedna implementácia v konkrétnom vývoji systému môže importovať daný stroj.

  Z hľadiska viditeľnosti je IMPORTS podobný INCLUDES, líši sa však v tom, že (abstraktné) premenné importovaného stroja sú v operáciách importujúcej implementácie viditeľné iba v invariantoch cyklov. Francúzka škola B povoľuje v implementácii aj klauzulu EXTENDS vo význame IMPORTS s automatickým zavedením všetkých operácií stroja. Podobne ako u INCLUDES aj tu možno použiť premenovanie.

Prehľad sprístupnenia poskytovaného jednotlivými klauzulami na úrovni abstraktných strojov poskytuje tabuľka 4.22. V tabuľke predpokladáme, že stroj $M_2$ pristupuje k stroju $M_1$. To, že dané prvky z $M_1$ sú prístupné príslušnej klauzule $M_2$ je indikované uvedením prvého znaku názvu mechanizmu v danom políčku tabuľky. Stĺpec "INCL.EXT. param." znamená parametre strojov sprístupnených stroju $M_2$ pomocou INCLUDES alebo EXTENDS.

| $M_2$ / $M_1$ | INCL.EXT. param. | PROPERTIES | INVARIANT | OPERATIONS |
|---|---|---|---|---|
| parametre | | | U | U |
| množiny | S | S,U,I | S,U,I | S,U,I |
| konštanty | S | S,U,I | S,U,I | S,U,I |
| stavové premenné | | | U,I | S,U,I (iba čítanie) |
| operácie | | | | I |

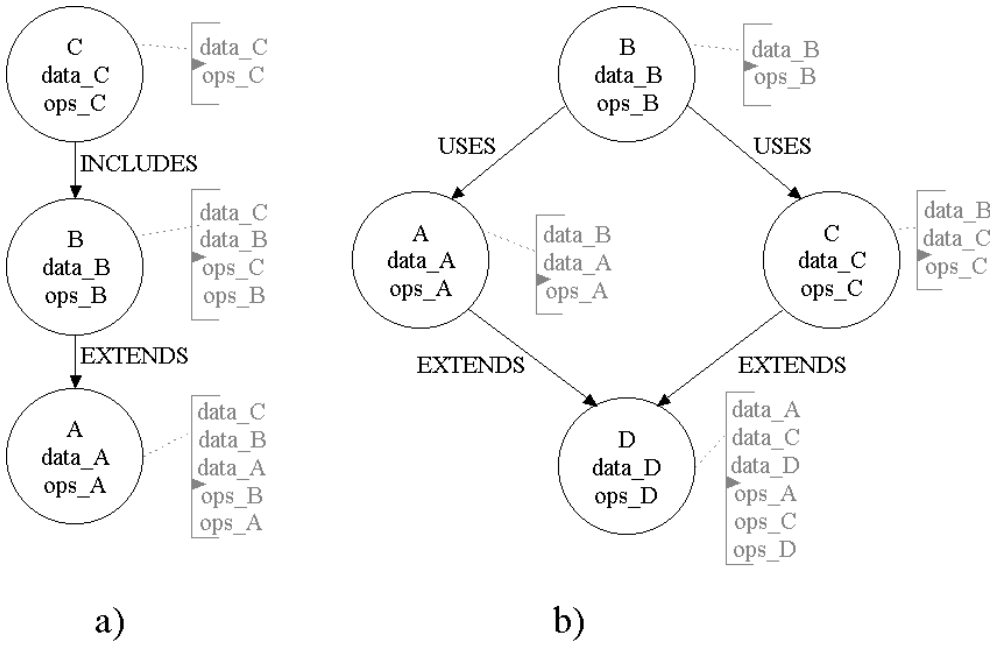
Tabuľka 4.22: Sprístupnenie na úrovni abstraktných strojov

USES, INCLUDES a EXTENDS sú mechanizmy *polovičného utajenia* (semihiding), čo znamená, že umožňujú čítať premenné sprístupneného komponentu operáciami pristupujúceho, ale nepovoľujú ich priamu aktualizáciu pristupujúcim komponentom. Tieto mechanizmy zabraňujú oddelenému zjemneniu sprístupneného komponentu nezávisle od pristupujúceho, pretože pristupujúci komponent závisí od reprezentácie údajov (od množiny premenných), ktorá je prítomná v sprístupnenom.

REFINES, SEES (v implementáciách) a IMPORTS sú mechanizmy *úplného utajenia* (full-hiding), pretože neumožňujú ani čítajúci prístup k premenným sprístupneného komponentu v operáciách pristupujúceho, mimo invariantu cyklu. Tieto mechanizmy podporujú nezávislé zjemnenie.

USES a SEES sú *intranzitívne* v zmysle, že ak $A$ používa (uses) $B$ a $B$ používa $C$, potom $A$ nemá prístup k žiadnemu prvku definovanému v $C$. Intranzitivita je tu zavedená kvôli úlohe týchto mechanizmov podporovať zdieľaný prístup ku komponentom. Konkrétne, ak umožníme dvom komponentom $A$ a $B$ pristúpiť pomocou USES alebo SEES k tretiemu komponentu $C$ a ak $A$ a $B$ sú obsiahnuté (included) v komponente $D$, potom by pri absencii intranzitivity existovali dve kópie prvkov $C$ v komponente $D$.

Na ilustráciu uvedieme dva príklady, objasňujúce použitie a efekt mechaniz-




Obr. 4.3: Príklady použitia kompozičných mechanizmov

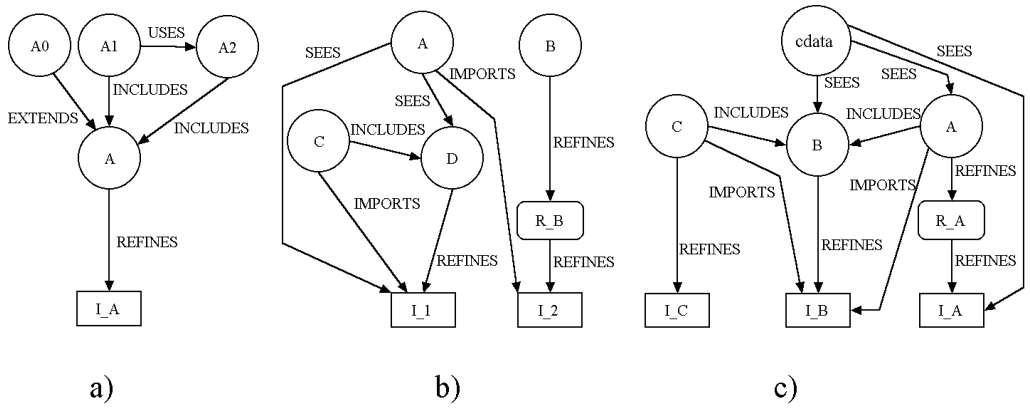
mov:

$A$ EXTENDS $B$; $B$ INCLUDES $C$ (obrázok 4.3 a): $A$ obsahuje všetky premenné, ktoré sú explicitne deklarované v $A$, plus premenné deklarované v $B$ a $C$. Podobne pre ostatné údajové prvky. Avšak operácie $A$ budú iba tie, ktoré sú explicitne deklarované v $A$ alebo v $B$.

$A$ USES $B$; $C$ USES $B$; $D$ EXTENDS $A$,$C$ (obrázok 4.3 b): $D$ obsahuje údajové prvky explicitne deklarované v $D$, $A$ alebo $C$, ale nie v $B$. Operácie $D$ sú tiež iba tie, ktoré sú explicitne deklarované v $D$, $A$ alebo $C$.

### 4.7.1 Stratégie vývoja v B

Medzi dekompozíciou systému na úrovni abstraktnej špecifikácie a na úrovni implementácie nemusí byť žiaden priamy vzťah. Na základe toho aký vzťah




Obr. 4.4: Príklady stratégií vývoja v B: monolitický (a), samostatná dekompozícia (b), nepretržitosť štruktúry (c)

medzi nimi existuje rozlišujeme 3 stratégie vývoja v B (obrázok 4.4)[15]:

- *Monolitický prístup* (monolithic approach) - na úrovni implementácie je len jeden systém (jedna implementácia), ktorý importuje množinu abstraktných strojov, tvoriacich systém na úrovni abstraktnej špecifikácie. Na implementačnej úrovni teda neexistuje dekompozícia, preto možno pri vytváraní špecifikácie použiť všetky kompozičné mechanizmy.

- *Samostatná dekompozícia* (separate decomposition) - dekompozícia na úrovni abstraktnej špecifikácie a na úrovni implementácie spolu priamo nesúvisia, počas vývoja sa objavujú nové subsystémy. O úplnej nezávislosti týchto dvoch dekompozícii možno však uvažovať len v prípade, ak na úrovni špecifikácie sú použité len mechanizmy úplného utajenia.

- *Nepretržitosť štruktúry* (continuity of structure) - dekompozícia implementácie kopíruje dekompozíciu abstraktnej špecifikácie. Každý komponent abstraktnej špecifikácie je samostatne zjemnený až do vykonateľného kódu. Ak je nejaký abstraktný stroj $A$ (pomocou INCLUDES) obsiahnutý v inom stroji $B$, potom implementácia $B$ väčšinou importuje stroj $A$. To spôsobí, že všetky množiny a konštanty z $A$ budú dva

[15]na obrázku sú kružnicami znázornené abstraktné stroje, oválmi zjemnenia a obdĺžnikmi implementácie

krát viditeľné v implementácii $B$ (raz cez klauzulu IMPORTS a druhý krát cez INCLUDES/EXTENDS). Aby sme predišli takejto situácii, vyberieme všetky množiny a konštanty z $A$ do samostatného stroja[16], ktorý bude videný strojmi $A$, $B$ aj implementáciou stroja $B$.

## 4.8 Formalizácia diagramatických modelov

Aj keď je možné priamo, na základe analýzy požiadaviek, zostaviť formálnu špecifikáciu systému, v praxi sa často najprv zostaví analytický diagramatický model, pozostávajúci z množiny diagramov, popisujúcich statické, dynamické a funkčné vlastnosti systému.

V tejto časti uvedieme postupy transformácie statických a dynamických modelov, ktoré sa využívajú pri objektovo orientovanom návrhu metódou OMT (Object Modeling Technique). Konkrétne pôjde o objektový diagram v prípade statického údajového modelu a o stavový diagram v prípade dynamického modelu. Podobné postupy sa používajú aj na transformáciu z diagramov jazyka UML [20].

### 4.8.1 Formalizácia statického údajového modelu

Objektový diagram pozostáva z pomenovaných obdĺžnikov, reprezentujúcich jednotlivé entity - triedy objektov. Každá trieda má uvedené meno a zoznam atribútov spolu s ich typmi. Vzťahy medzi triedami sú znázornené čiarami spájajúcimi príslušné obdĺžniky.

Proces mapovania objektového modelu do jazyka B-AMN pozostáva z týchto krokov:

1. Identifikácia rodiny typov entít v údajovom modeli. Rodina typov entít je taká skupina typov, ktoré sú podtypmi nejakého typu $T$, ktorý nemá žiaden nadradený typ.

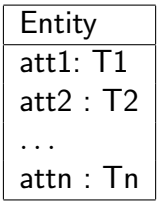
2. Identifikácia prístupových ciest medzi rodinami entít, potrebných pre ich operácie a atribúty.

3. Na základe identifikovaných ciest vypracovať orientovaný acyklický graf, ktorého vrcholmi sú rodiny a hrany vyjadrujú vzťahy typu USES a SEES medzi rodinami.

[16]Na obrázku 4.4 c) je takýto stroj označený "cdata". Jeho implementácia nie je znázornená z priestorových dôvodov.

4. Definovať abstraktné stroje pre každú rodinu podľa postupu popísaného nižšie a zahrnúť stroje do iných strojov podľa vzťahov identifikovaných v kroku 3.

Postup definovania abstraktného stroja pre jednoduchý prípad entity (triedy) bez podtypov (zdedených tried), vrátane operácie vytvárajúcej novú inštanciu entity, je na obrázku 4.5.




a)

MACHINE *Entity*
SETS $ENTITY$
VARIABLES $entities, att1, att2, \ldots, attn$
INVARIANT $entities \subseteq ENTITY \wedge att1 \in entities \rightarrow T1$
$\wedge \ldots \wedge attn \in entities \rightarrow Tn \wedge \ldots$
INITIALISATION $entities := \varnothing \parallel att1 := \varnothing \parallel \ldots \parallel attn := \varnothing$
OPERATIONS
ee ← createEntity(atv1, ..., atvn) $\widehat{=}$
PRE $atv1 \in T1 \wedge \ldots \wedge atvn \in Tn \wedge entities \neq ENTITY$
THEN ANY $oo$ WHERE $oo \in ENTITY - entities$ THEN
$ee := oo \parallel entities := entities \cup \{oo\} \parallel$
$att1(oo) := atv1 \parallel \ldots \parallel attn(oo) := atvn$
END
END
END

b)

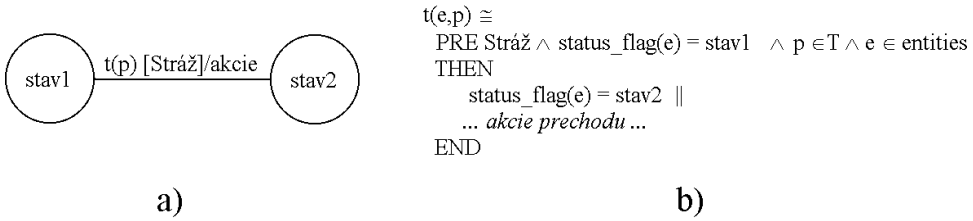
Obr. 4.5: Trieda *Entity* v objektovom diagrame (a) a zodpovedajúci abstraktný stroj (b)

Entitu (triedu) *Entity* v B teda vyjadríme ako abstraktný stroj, ktorý v sebe zahŕňa množiny všetkých možných inštancií (množina identity objektov $ENTITY$) a všetkých existujúcich inštancií triedy (množina *entities*). Operácia vytvárajúca novú inštanciu entity (create_entity) vyberie ľubovoľnú ešte nevybranú (neinicializovanú) inštanciu z $ENTITY$, priradí jej hodnoty atribútov a zaradí ju do množiny *entities*.

Ak sú medzi entitami v modeli väzby (vzťahy), využívame na sprístupnenie príslušných strojov v B-AMN mechanizmy SEES alebo USES. SEES v prípade, že iba potrebujeme použiť množinu identity objektov (ENTITY) jedného stroja v invariante druhého. Ak potrebujeme v invariante či operáciách použiť aj množinu existujúcich objektov (entities) iného stroja, použijeme USES. Dedičnosť modelujeme mechanizmom EXTENDS, agregáciu mechanizmom INCLUDES.

### 4.8.2 Formalizácia dynamického modelu

Stavový diagram pozostáva z vrcholov, predstavujúcich stavy systému, ktoré sú poprepájané hranami, znázorňujúcimi stavové prechody. Prechodom je pridelená strážna podmienka a (voliteľne) tiež akcie, ktoré sa vykonajú pri jeho realizácii.




Obr. 4.6: Fragment stavového diagramu (a) a prepis stavového prechodu $t(p)$ do operácie B-AMN (b)

Proces transformácie stavového diagramu do jazyka B-AMN, vhodný aj pre veľký počet stavov, je nasledovný:

1. Vytvoriť abstraktný stroj *Entity* pre každú rodinu typov entít v údajovom modeli (ako v predchádzajúcom postupe) a premennú $status\_flag_i$, definovanú ako funkciu $status\_flag_i : \ entities \rightarrow STATUS\_SET_i$. Táto premenná bude zaznamenávať stav $i$-teho faktora[17] stavového diagramu pre *Entity*. $STATUS\_SET_i$ je množina (reprezentantov) stavov v $i$-tom faktore.[18]

2. Vytvoriť operáciu stroja *Entity* pre každý prechod a aktivitu v každom stavovom diagrame, ktorá vykoná akcie prechodu a zmení stav zmenou hodnoty príslušnej (alebo príslušných, ak sa prechod vyskytuje vo viacerých diagramoch) $status\_flag_i$ premennej (premenných). Vytvorenie operácie znázorňuje obrázok 4.6.
   Operácia vytvorenia novej inštancie entity (create_entity) musí každú z funkcií $status\_flag_i$ inicializovať, tak aby jej hodnota reprezentovala počiatočný stav daného faktora diagramu.

[17]Faktor $H$ grafu $G$ je taký podgraf $G$, ktorý má množinu vrcholov zhodnú s $G$ (formálne $V(H) = V(G)$).

[18]V jednoduchších prípadoch, keď máme len jeden diagram, si vystačíme s jedinou množinou (reprezentantov) stavov $STATUS\_SET$ a teda aj s jedinou premennou $status\_flag : \ entities \rightarrow STATUS\_SET$.

3. Vyjadrenie synchronizácie medzi subsystémami operáciami, ktoré volajú operácie strojov reprezentujúcich subsystémy a to s použitím operátora "||".

# Zoznam skratiek

V nasledujúcej tabuľke je uvedený zoznam skratiek použitých v texte práce. V zátvorke za slovenským významom je u väčšiny skratiek uvedený aj anglický význam.

| **Skratka** | **Význam** |
|---|---|
| B | B-Metóda (B-Method) |
| B-AMN | B - notácia abstraktných strojov (B - Abstract Machine Notation) |
| CBPN | kapacitne obmedzená Petriho sieť (Capacity Bounded Petri Net) |
| EPN | hodnotiaca Petriho sieť (Evaluative Petri Net) |
| GPN | zovšeobecnená Petriho sieť (Generalised Petri Net) |
| GS | zovšeobecnená substitúcia (Generalized Substitution) |
| GSL | jazyk zovšeobecnenej substitúcie (Generalized Substitution Language) |
| LGC | Jazyk strážených príkazov (Language of Guarded Commands) |
| PS | Petriho sieť (Petri Net) |

# Literatúra


[1] Abrial, J. R.: *The B Book: Assigning programs to Meaning*. Cambridge University Press, 1996.

[2] Abrial, J. R., Lecomte, T., Mussat L.: *Event B Reference Manual*. Methodologies and Technologies for Industrial Strength Systems Engineering, Project report, Jún 2001.

[3] Bača, J., Hudák, Š.: *De/compositional Time Reachability Analysis*. Proceedings of EMES'2001 Oradea, Rumunsko, 24.-26. Máj 2001, pp.60-65.

[4] Bouziane, Z.: *A Primitive Recursive Algorithm for the General Petri Net Reachability Problem*. Technical report n3404, Institut National de Recherche en Informatique et en Automatique, Rennes, Francúzko, Apríl 1998, 14 pp.

[5] Bowen, B.P., Hinchey, M.: *Ten Commandments of Formal methods*. IEEE Computer, Vol. 28, No. 4, Apríl 1995, pp.56-63.

[6] Clarke, E.M., Wing J.M. : *Formal methods: State of art and future directions*. ACM Computing Surveys, Vol. 28, No. 4, December 1996, pp. 626–643.

[7] Dick, J., Lano, K.: *Development of Concurrent Systems in B AMN*, Dept. of Computing, Imperial College, 1995, 27 pp.

[8] Dijkstra, E.W.: *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, 1976, 217 pp, ISBN 0-13-215871-X.

[9] Frappier, M., Habrias, H.: *Software Specification Methods: An Overview Using a Case Study*. Springer-Verlag, 2000, 312 pp, ISBN 1-85233-353-7.

[10] Hudák, Š.: *Rozšírenia Petriho Sietí*. Habilitačná práca, VŠT EF Košice, 1980, 107 pp.

[11] Hudák, Š.: *De/compositional Reachability Analysis.* Journal of Electrical Engineering, Vol. 45, No. 11, 1994 pp.424-431.

[12] Hudák, Š.: *New Approach to Solving the Reachability Problem: on the way to methodology of the analysis of time-critical systems.* DrSc thesis, Národná univerzita T. Ševčenka, Kijev, Ukrajina, 1995, 270 pp.

[13] Hudák, Š.: *Time Interval Semantics of TB nets.* Proceedings of RSEE'96 Oradea, 30.-31., Máj 1996, pp.294-310.

[14] Hudák, Š.: *Reachability Analysis of Time-Critical Systems.* Mathematical Methods in Cybernetics, Communications of the International Solomon University, Kijev, Vol. 3, No. 4, Október 2000, pp.49-78.

[15] Hudák, Š., Grofčík J.: *An Environment for Design and Analysis of Time-Critical Systems.* Proceedings of EMES'2001, Oradea, 24.-26. Máj 2001, pp.66-75.

[16] Jensen, K.: *An Introduction to the Theoretical Aspects of Coloured Petri Nets.* A Decade of Concurrency, Lecture Notes in Computer Science vol. 803. Springer-Verlag 1994, pp. 230-272.

[17] Jensen, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Springer-Verlag, 1997, ISBN 3-540-60943-1.

[18] Jensen, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Springer-Verlag, 1997, ISBN 3-540-58276-2.

[19] Jensen, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use.* Springer-Verlag, 1997, ISBN 3-540-62867-3.

[20] Laleau, R., Polack, F.: *Coming and Going from UML to B: A Proposal to Support Traceability in Rigorous IS Development.* Proceedings of the 2nd International Conference of B and Z Users, Grenoble, Francúzko, 23.-25. Január 2002, pp. 517-534.

[21] Lano K.: *Specifying Reactive Systems in B AMN.* In J.P. Bowen, M.G. Hinchey and D. Till, editors, ZUM'97, volume 1212 of LNCS, Springer Verlag, 1997, pp. 242275.

[22] Lano, K.: *The B Language and Method: A Guide to Practical Formal Development*. Springer-Verlag London, 1996, 232 pp, ISBN 3-540-76033-4.

[23] Murata, T.: *Petri Nets: Properties, Analysis and Applications*. Proceedings of the IEEE, Vol. 77, No. 4, Apríl 1989, pp.541-579.

[24] Peterson, J.: *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Engelwood Cliffs, New Jersey, 1981.

[25] Petri, C.A.: *Kommunikation mit Automaten*. Dizertačná práca, Institut für Instrimentelle Mathematik, University Bonn, 1962.

[26] Reisig, W.: *Petri Nets - an Introduction*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

[27] Šimoňák, S.: *Formálne metódy špecifikácie a analýzy časovo-kritických systémov*. Písomná práca k dizertačnej skúške, KPI FEI TU Košice, 2000, 74 pp.

[28] Teliopoulos, K.G.: *Časová analýza časovo-kritických systémov*. Písomná práca k dizertačnej skúške, KPI FEI TU Košice, 1998, 46 pp.

[29] *The B-Method Reference Manual*. Referenčný manuál B-AMN pre B-Toolkit od firmy B-Core. Dostupný z: http://www.b-core.com/downloading.html.

[30] *The B Language Reference Manual*. Referenčný manuál B-AMN pre Atelier B a B4free od firmy ClearSy. Dostupný z: http://www.b4free.com/public/resources.php.

[31] *Atelier-B Web server*. http://www.atelierb.eu/index_en.html.

[32] *Web stránka nástroja B4free*. http://www.b4free.com.

[33] *Web stránka firmy B-Core*. http://www.b-core.com.

[34] *Web stránka firmy ClearSy*. http://www.clearsy.com.

[35] *Web stránka nástroja CPN Tools*. http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[36] *World of Petri Nets*. Homepage of the International Petri Net Community, http://www.daimi.au.dk/PetriNets.