M. Malík; Zdeněk Benda
Počítačová simulace

Pokroky matematiky, fyziky a astronomie, Vol. 29 (1984), No. 1, 1--29

Persistent URL: http://dml.cz/dmlcz/139021

POKROKY MATEMATIKY, FYZIKY A ASTRONOMIE, ROČNÍK XXIX (1984) ČÍSLO 1

Počítačová simulace

Marek Malík, Praha, Zdeněk Benda, Brno

Samočinné počítače se na počátku své éry používaly ve dvou oblastech. Jednak to bylo provádění vědeckotechnických výpočtů — zejména v souvislosti s řešením různých numerických úloh, jednak šlo o zpracování tzv. hromadných dat, obvykle různých agend správního a organizačního charakteru. V těchto dvou „klasických" oblastech se počítače s úspěchem používají dodnes. Patrně při tom nikdo nepochybuje, že je vhodné používat služeb samočinných počítačů. (Je sice pravda, že počítačové zpracování hromadných dat umožnilo vznik různých agend v rámci automatizovaných systémů řízení a že si lze u mnoha z těchto agend položit otázku, zda mají vůbec nějaký racionální smysl, ale to je věc z principu jiná, která nesouvisí s vlastními výhodami použití počítačů pro řešení „rozumných" úloh z oboru hromadných dat.)

Nepoměrně menší je shoda názorů na výhody přinášené použitím počítačů v „neklasických" oblastech podle nověji vzniklých (a vznikajících) metodických postupů. Je možno se setkat s krajním názorem, že prakticky všechny „netradiční" aplikace počítačů jsou naprostý podvod a mámení finančních prostředků ze státního rozpočtu, a naopak jsou jiné hlasy, které považují nové aplikační sféry počítačů za jediný možný prostředek ke spáse světa, a to jak reálného světa kolem nás, tak třebas i abstraktního světa klasické matematiky. „Pravda" bude patrně — jak to ostatně bývá — někde uprostřed. Jsou zcela jistě „nové" metody a metodiky používání počítačů schopné poskytovat výsledky dosud vůbec nezískatelné či získatelné jen s krajními obtížemi. Jednou z takových výzkumných metod je i počítačová simulace.

Následující článek vznikl s úmyslem podat čtenářům PMFA představu o tom, co to vůbec počítačová simulace je. Článek je volně napsán podle referátu, který autoři přednesli na semináři Sofsem v roce 1980.

1. OBECNĚ O METODĚ MODELOVÁNÍ

1.1. Intuitivní představy

Zamysleme se nejprve nad pojmy „*modelování*" a „*simulace*" bez všech nároků na exaktnost a řekněme si (nejprve samozřejmě jen velmi přibližně), co si budeme pod těmito pojmy představovat.

Jak již sám název napovídá, je základem výzkumné metody modelování tvorba rozličných modelů. Pojem model zde chápeme poněkud obecněji než v obvyklém hovorovém slova smyslu. Jde v podstatě o to, že v situaci, kdy máme zkoumat nějaký úsek reálného světa, zkoumáme úsek jiný (z nějakého hlediska pro nás výhodnější), který má takové vlastnosti, že získané poznatky lze snadno aplikovat na úsek původní.

Značně zhruba řečeno, může jít například o situaci, kdy zkoumáme reakci lidského organismu na nákazu jistým druhem mikrobů a modelem člověka je krysa.

Důvody, které nás vedou k používání modelů, jsou rozličné.

Mohou to být například důvody ekonomické – pořizovací cena modelu i cena na něm prováděných pokusů je často neporovnatelně nižší než cena originálu a pokusů na něm prováděných (uveďme například hydromechanické pokusy na několika různých maketách dosud nepostavené přehrady prováděné za účelem stanovení optimálního tvaru její některé části, pokusy s maketami automobilů v aerodynamickém tunelu a podobně).

Použití modelu vede rovněž velmi často k časovému zisku – pokus na modelu může trvat podstatně kratší dobu než pokus na originálu. To vyniká zejména tehdy, použijeme-li k tvorbě modelu samočinný počítač. Modelujeme-li například pomocí samočinného počítače zemědělský systém rostlinné výroby za účelem výběru nejlepšího (to jest třebas nejlacinějšího) způsobu rozvozu přirozeného hnojiva z několika možností daných technologií, může být výsledek (včetně přípravné programátorské práce) získán během několika dní, zatímco pokusy na originálu (nehledě na jejich ekonomickou náročnost) by trvaly mnoho měsíců.

V neposlední řadě nás k používání modelů vedou i důvody řekněme principiální, když modelujeme nikoli objekt z reálného světa kolem nás, ale jen hypotetickou představu o nějaké části či situaci reálného světa. Patří sem například (nepočítáme-li takové modely jako strategické studie třetí světové války) modelové výpočty přistávání nově projektované meziplanetární sondy na Marsu, jejichž výsledky vedou teprve k upřesňování projektu sondy.

Hned ale musíme poznamenat, že uvedený výčet důvodů používání modelů je do značné míry umělý – ve valné většině případů se setkáváme s kombinací mnoha a mnoha důvodů (i takových, které jsme neuvedli – například etických, které nám brání používat k pokusům s kancerogenními látkami lidských dobrovolníků a vedou k již vzpomenutému modelování člověka krysou).

Po uvedeném zdůvodnění proč modelovat se vraťme opět k intuitivním představám o modelování a pokusme se poněkud přesněji a obsažněji než výše vymezit význam pojmu model.

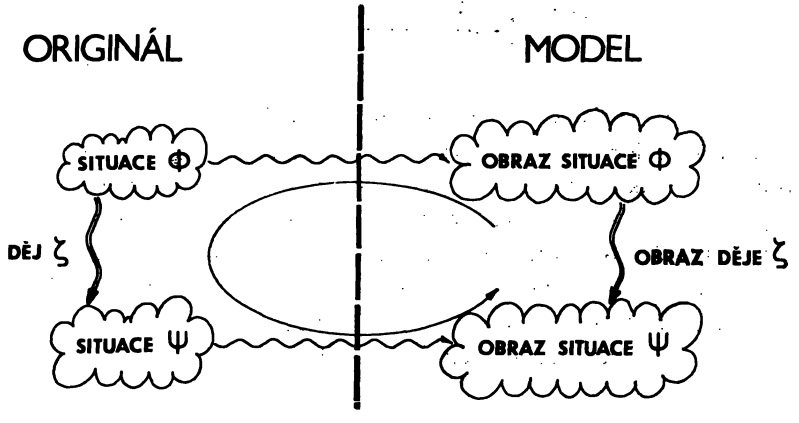
Především stojí za povšimnutí, že jak modelovaný originál, tak jeho model jsou celky s jistou vnitřní logikou, která umožňuje, aby v těchto celcích vznikaly různé situace. K označování takových celků se většinou používá pojem *systém*. Význam pojmu systém byl (jak se ještě dále zmíníme) v literatuře mnohokrát diskutován a definován. Pokoušet se zde o další definici systému by bylo jistě neúčelné; omezíme se jen na úmluvu (pro potřeby našich intuitivních představ zcela postačující), že systémem budeme rozumět libovolnou logicky vymezenou část reálného světa nebo hypotetickou představu o takové části. Logickým vymezením rozumíme asi tolik, že jsou v systému rozlišitelné různé situace, že za nedílnou část systému považujeme i děje, které jej převádějí ze situace do situace a že je soubor situací systému uzavřen vůči všem jeho dějům (jako zobrazením ze situací do situací).

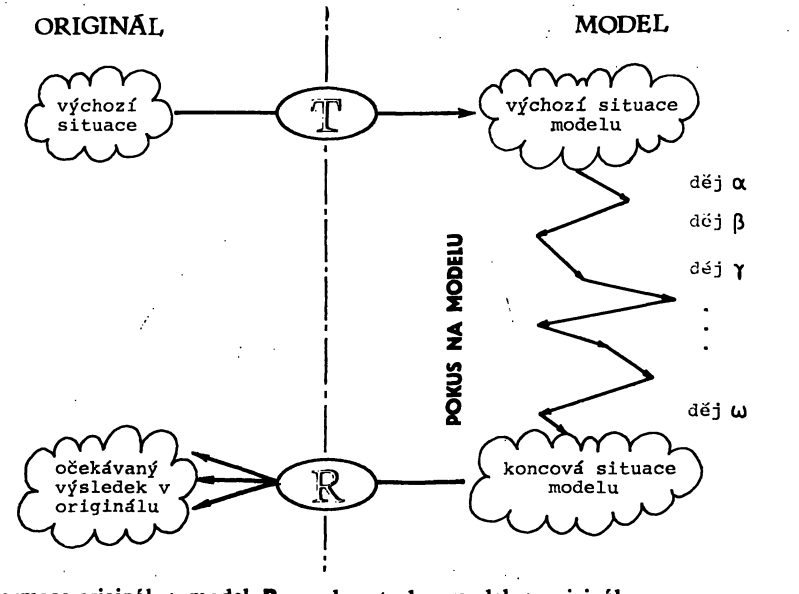
Po této úmluvě o systému lze prohlásit, že originál i jeho model jsou systémy, které jsou si jistým způsobem podobné. Přesněji, že každé situaci v originálu odpovídá jednoznačně nějaká situace v modelu a podobně každému ději v originálu odpovídá jednoznačně nějaký děj v modelu, přičemž je zachována i logická souvislost mezi ději a situacemi, což lze znázornit schématem na obr. 1.

Zobrazení situací a dějů originálního systému v jeho modelu nemusí být zdaleka prosté. To záleží na tom, jakou přesnost pohledu na originální systém vyžaduje konkrétní modelovací úloha – podrobněji na tom, mezi kterými situacemi a ději v originálu je třeba rozlišovat a mezi kterými nikoli. Ze situace v modelu pak samozřejmě není možno rekonstruovat přímo jedinou vzorovou situaci v originálu, pouze celou skupinu vzorových situací (mezi nimiž ovšem nemíníme momentálně uvažovat nějaký rozdíl).

Postup při použití metody modelování lze pak charakterizovat tak, že na základě chování modelu za jistých podmínek usuzujeme na chování originálu za analogických (vzorových) podmínek. Přesněji, chceme-li zjistit situaci, do níž se originál dostane po nějaké sérii dějů, zjistíme situaci, do níž se dostane jeho model po odpovídající sérii vlastních dějů a z té rekonstruujeme očekávaný výsledek v originálu. Schematicky je toto znázorněno na obr. 2.

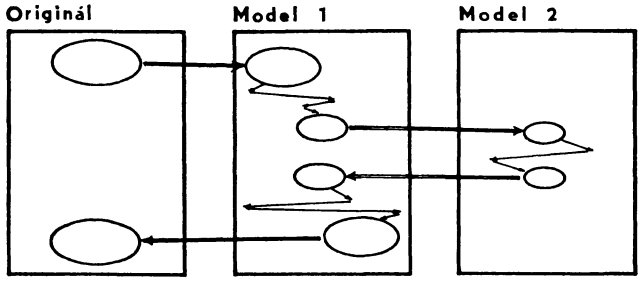
Obr. 1





Obr. 2. **T** — transformace originál → model, **R** — rekonstrukce model → originál.

Ze systémového pojetí originálu i modelu vyplývá, že je možné, aby systém modelu byl sám vzorem pro modelový systém vyšší úrovně. Na takovém principu lze uvažovat i realizovat modelové projekty využívající modelů vyšších úrovní. (Příkladem může být např. studie, v níž je nějaký systém modelován fyzikálním experimentem, jehož počáteční podmínky jsou nastaveny podle výsledku předem provedeného modelovacího výpočtu na samočinném počítači.) Obr. 3.

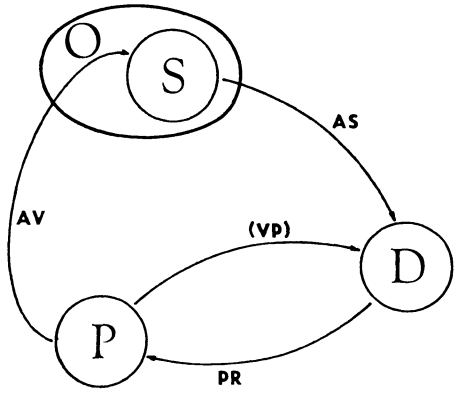


Obr. 3.

Již padla výše několikrát zmínka, že k vytváření modelu je možno využít samočinný počítač. Modelem je potom výpočet modelujícího programu (přesněji systém výpočtu programu). Takový postup v modelování na samočinném počítači, kdy v modelujícím programu zobrazujeme i časové vlastnosti dějů originálního systému (trvání jednotlivých dějů, odstupy mezi změnami a podobně), nazýváme *počítačovou simulací.*

Poznamenejme k tomu jen na okraj, že vymezení pojmů simulace a modelování je v literatuře často diskutováno; jako v případě mnoha jiných rozšířenějších pojmů lze i pro simulaci nalézt různé „definice" mnohdy se diametrálně lišící – jmenujme alespoň [3].

Protože v našem článku půjde hlavně o simulační modelování, povšimněme si ještě obecného postupu vytvoření simulačního programu. Tento postup může být ilustrován schématem na obr. 4.



Obr. 4. **AS** — analýza systému, **PR** — programování, **VP** — verifikace programu, **AV** — aplikace výsledků.

Na sledovaném objektu **O** nejprve vymezíme podle zvolené rozlišovací úrovně modelový systém **S** (to představuje především určení, co nás bude na objektu **O** zajímat a co ne). Ve druhé fázi přistoupíme k analýze chování systému **S**, což vede k vytvoření popisu jeho činnosti **D**. Pokud je to možné, je výhodné, aby popis **D** byl nějakým způsobem formalizován – použitý aparát je však silně závislý na povaze systému **S** (tedy vlastně na povaze objektu **O**) a na požadavcích simulační úlohy. Jako často používané prostředky popisu zde jmenujme alespoň diferenciální, diferenční a algebraické rovnice, operátorové přenosy, bloková (regulační) schémata, grafové metody (jako metodu signálových toků, Petriho a vyhodnocovacích sítí) a podobně. Kromě toho se používají nejrůznější další prostředky včetně slovního popisu (i značně neformalizovaného). Prakticky ve většině případů lze k jednomu sledovanému systému **S** vytvořit různé typy popisů **D**.

Dalším krokem je vlastní programování. Struktura simulačního programu **P** je určena převážně tvarem popisu **D** a použitým programovacím jazykem. Programování může být značně usnadněno, jsou-li formální prostředky popisu **D** a použitý programovací jazyk blízké ve smyslu způsobu vyjadřování. Činí se i pokusy o formalizované popisové prostředky s přesně definovanou syntaxí a sémantikou, které by umožňovaly automatický (tedy počítačem prováděný) převod formálního popisu do modelujícího programu v některém simulačním (nebo pro simulaci vhodném) programovacím jazyce – [26, 27]. Některé vyšší programovací jazyky lze dokonce přímo používat jako jazyky deskripční – to jest umožňující dostatečně přehledný zápis formálního popisu zkoumaného systému. Zkušenost však ukazuje, že ani v těchto případech neodpadá vlastní programování (vytváření modelujícího programu na základě popisu vlastností originálního systému). Použije-li se totiž pro simulační výpočet program přímo v původním tvaru formálního popisu systému, velice obtížně se ladí. (Přeprogramování formálního popisu je zde ale relativně snadné.)

Výsledky získané v simulačních experimentech s modelem **P** je pak možné interpretovat v modelovaném systému **S**. Praktické zkušenosti ukazují, že celý cyklus se zpravidla opakuje (nezřídka mnohokrát), dokud se nedosáhne vyhovujícího vztahu mezi věrností zobrazení modelovaného systému v programu na jedné straně a komplikovaností programu (počítaje v to jeho nároky na strojový čas a operační paměť počítače) na straně druhé.

Teoreticky lze uvažovat ještě verifikaci simulačního programu, to jest formálně provedený důkaz, že navržený program **P** odpovídá popisu **D**. Ovšem značná komplikovanost aparátu, který je k takovému důkazu zapotřebí, způsobuje, že se obvykle simulační programy neverifikují (ostatně se to nedělá prakticky u žádných komplikovanějších programů). Hlavní problémy vytváření simulačních programů však nejsou téměř nikdy v chybách při programování (které by verifikace programu mohla nalézt), ale v analýze systému, resp. ve vymezení modelovaného systému na sledovaném objektu. Formálně dokazovat správnost formalizovaného popisu zkoumaného systému je ale principiálně nemožné – znamenalo by to vlastně dokazovat, že jsou správné naše (krajně neformalizované) představy o zkoumaném systému a že jsme jeho jednotlivé vlastnosti a vazby mezi nimi správně pochopili.

1.2. Formalizace pojmů

Po intuitivním úvodu se nyní postavme na zcela opačné stanovisko a zamysleme se nad modelováním z pozice exaktní matematiky.

Chápeme-li vazby mezi originálem a jeho modelem jako vztahy mezi dvěma systémy, musíme nutně před formálním popisem a rozborem těchto vazeb exaktně formalizovat pojem systém. Exaktní formalizace pojmu systém byla opakovaně provedena různými autory. V literatuře je možno nalézt celou řadu definic systému; mnohdy jsou to definice, které se vzájemně velmi podstatně liší. Zde uvedeme jednu z nejpřehlednějších definic, která vychází z analogie s konečnými automaty [21].

Nejprve zavedeme několik pomocných pojmů.

Definice. Buď A neprázdná množina, E množina reálných čísel. Zobrazení $\omega: E' \subseteq E \to A$ nazveme E segmentem (nebo jenom segmentem) množiny A, je-li jeho definiční obor dom (ω) průnikem množiny E s intervalem reálných čísel.

Buďte ω_1, ω_2 dva segmenty množiny A takové, že

$$\sup(\mathrm{dom}(\omega_1)) \leqq \inf(\mathrm{dom}(\omega_2))$$

a

$$\neg\,(\exists x \in E)\,(\sup(\mathrm{dom}(\omega_1)) < x < \inf(\mathrm{dom}(\omega_2)));$$

potom o těchto segmentech řekneme, že na sebe navazují, a segment ω množiny A takový, že

$$\mathrm{dom}(\omega) = \mathrm{dom}(\omega_1) \cup \mathrm{dom}(\omega_2) \cup (\{\sup(\mathrm{dom}(\omega_1)), \inf(\mathrm{dom}(\omega_2))\} \cap E),$$

$$(\forall t)\,(t \in \mathrm{dom}(\omega_i) \setminus \{\sup(\mathrm{dom}(\omega_1)), \inf(\mathrm{dom}(\omega_2))\} \to \omega(t) = \omega_i(t),\ i = 1, 2),$$

nazveme kompozicí segmentů ω_1, ω_2.

Kompozice segmentů není uvedenou definicí určena jednoznačně. Můžeme však její hodnoty na množině

$$\{\sup(\mathrm{dom}(\omega_1)), \inf(\mathrm{dom}(\omega_2))\} \cap E$$

považovat za definované nějakým standardním způsobem. Pak lze na množině právě všech E segmentů množiny A uvažovat asociativní binární operátor *Comp* takový, že pro každé dva segmenty ω_1 a ω_2 je *Comp* (ω_1, ω_2) definováno právě tehdy, když ω_1 a ω_2 na sebe navazují, a v takovém případě je *Comp* (ω_1, ω_2) kompozicí segmentů ω_1, ω_2.

A nyní již můžeme přistoupit k definici systému.

Definice. Systémem $\mathscr{S}$ nazveme uspořádanou sedmici $\langle \mathbf{T}, X, \Omega, \mathbf{Q}, Y, \delta, \lambda \rangle$, kde

$\mathbf{T}$ je množina reálných čísel — časová množina systému,

X, resp. Y jsou neprázdné množiny vstupních, resp. výstupních hodnot systému,

Ω je množina $\mathbf{T}$ segmentů množiny X — množina vstupních segmentů systému,

$\mathbf{Q}$ je množina stavů systému,

δ je přechodová funkce systému,

λ je výstupní funkce systému,

platí-li:

1. Množina Ω je uzavřena na kompozici.
2. δ je zobrazení $\mathbf{Q} \times \Omega \to \mathbf{Q}$ a λ je zobrazení $\mathbf{Q} \to Y$.
3. Pro každé segmenty $\omega_1, \omega_2 \in \Omega$, které na sebe navazují a pro každé $q \in \mathbf{Q}$ platí

$\delta(q, Comp(\omega_1, \omega_2)) = \delta(\delta(q, \omega_1), \omega_2)$.

Práci takto definovaného systému lze popsat takto:

Nachází-li se systém $\mathscr{S}$ v čase $t_0 \in \mathbf{T}$ ve stavu $q_0 \in \mathbf{Q}$ a obdrží-li dále v okamžicích $\langle t_0, t_1 \rangle \cap \mathbf{T}$, kde $t_1 \in \mathbf{T}$, vstupní segment $\omega_0 \in \Omega$ ($\mathrm{dom}(\omega_0) = \langle t_0, t_1 \rangle \cap \mathbf{T}$), pak v okamžiku t_1 bude ve stavu $q_1 = \delta(q_0, \omega_0)$ a vydá výstupní hodnotu $\lambda(q_1)$.

Na základě definice systému můžeme nyní definovat i vztah modelu mezi dvěma systémy.

Definice. O systému

$\mathscr{M} = \langle \mathbf{T}_m, X_m, \Omega_m, \mathbf{Q}_m, Y_m, \delta_m, \lambda_m \rangle$

řekneme, že je izomorfním modelem systému

$\mathscr{S} = \langle \mathbf{T}_s, X_s, \Omega_s, \mathbf{Q}_s, Y_s, \delta_s, \lambda_s \rangle$,

jestliže existují zobrazení

$\tau : \mathbf{T}_s \to \mathbf{T}_m$, $\alpha : X_s \to X_m$, $\varphi : \Omega_s \to \Omega_m$, $\psi : \mathbf{Q}_s \to \mathbf{Q}_m$, $\beta : Y_m \to Y_s$

taková, že:

1. τ je izomorfismus uspořádaných množin reálných čísel,
2. $(\forall \omega \in \Omega_s)\,(\mathrm{dom}(\varphi(\omega)) = \tau(\mathrm{dom}(\omega)) \,\&$
   $(\forall t \in \mathrm{dom}(\omega))\,(\varphi(\omega)(\tau(t)) = \alpha(\omega(t))))$,
3. $(\forall q \in \mathbf{Q}_s,\ \omega \in \Omega_s)\,(\lambda_s(\delta_s(q, \omega)) = \beta(\lambda_m(\delta_m(\psi(q), \varphi(\omega)))))$.

Neboli:

Vstupní segment modelovaného systému $\mathscr{S}$ se převádí „do řeči" modelu pomocí vstupního zobrazení α a časové transformace τ. Po nastavení do odpovídajícího počátečního stavu poskytuje model — po zpětném převodu zobrazením β — stejné výsledky jako originál.

Podaná definice je založena na požadavku „vnějšího" (vstupně-výstupního) vztahu mezi modelem a originálem. Pro vlastní vytváření modelu je ale podstatnější „vnitřní" podobnost mezi modelem a vzorem — to jest nejen podobnost vstupů a výstupů, ale i podobnost změn stavů. Znamená to v podstatě požadovat, aby proti podané definici navíc platilo*):

4. $(\forall q \in \mathbf{Q}_s,\ \omega \in \Omega_s)\,(\psi(\delta_s(q, \omega)) = \delta_m(\psi(q), \varphi(\omega)))$,
5. $(\forall q \in \mathbf{Q}_s)\,(\lambda_s(q) = \beta(\lambda_m(\psi(q))))$.

Jak jsme se však v úvodu zmínili, je podstatnou vlastností modelování to, že model se nemusí chovat přesně jako originál. Nemusíme proto požadovat naprostou shodu výsledků originálu a modelu:

Definice. O systému

$\mathscr{M} = \langle \mathbf{T}_m, X_m, \Omega_m, \mathbf{Q}_m, Y_m, \delta_m, \lambda_m \rangle$

řekneme, že je modelem systému

$\mathscr{S} = \langle \mathbf{T}_s, X_s, \Omega_s, \mathbf{Q}_s, Y_s, \delta_s, \lambda_s \rangle$,

jestliže existují zobrazení

$\tau : \mathbf{T}_s \to \mathbf{T}_m$, $\alpha : X_s \to X_m$, $\varphi : \Omega_s \to \Omega_m$, $\psi : \mathbf{Q}_s \to \mathbf{Q}_m$, $\gamma : Y_m \to \{Y;\ Y \subseteq Y_s\}$

taková, že

1*. τ je homomorfismus uspořádaných množin reálných čísel,

2*. $(\forall \omega \in \Omega_s)\,(\mathrm{dom}(\varphi(\omega)) = \tau(\mathrm{dom}(\omega)) \,\&$
   $(\forall t \in \mathrm{dom}(\omega))\,(\varphi(\omega)(\tau(t)) = \alpha(\omega(t))))$,

3*. $(\forall q \in \mathbf{Q}_s,\ \omega \in \Omega_s)\,(\lambda_s(\delta_s(q, \omega)) \in \gamma(\lambda_m(\delta_m(\psi(q), \varphi(\omega)))))$.

---

*) Bod 3. pak plyne z bodů 4. a 5.

Platí-li navíc

4*. $(\forall q \in Q_s, \omega \in \Omega_s)\ (\psi(\delta_s(q, \omega)) = \delta_m(\psi(q), \varphi(\omega)))$,

5*. $(\forall q \in Q_s)\ (\lambda_s(q) \in \gamma(\lambda_m(\psi(q))))$,

řekneme, že $\mathscr{M}$ je vnitřním modelem systému $\mathscr{S}$.

Na rozdíl od izomorfních modelů není tedy zde možná přesná rekonstruovatelnost výsledků originálního systému na základě výsledků modelu.

Poznamenejme ještě, že ač se na právě uvedenou teorii budeme občas odvolávat v dalších částech článku, nebudou tyto odvolávky (ostatně dosti řídké) nijak fundamentální, a že jsme proto pojmy z matematické teorie modelování prezentovali v jejich nejjednodušší podobě. Přesný teoretický základ dále zaváděných konstrukcí by byl podstatně komplikovanější.

2. PŘÍKLADY

V této kapitole si uvedeme zadání několika simulačních úloh na třech příkladech dosti jednoduchých reálných systémů. Příklady jsme zvolili tak, aby pokrývaly co nejširší okruh problémů a programových konstrukcí, se kterými je možno se setkat v simulačním programování. (Uvedené příklady ovšem samozřejmě nemohou úplně pokrýt problematiku simulace.) V dalším textu se budeme na tyto příklady odvolávat a ilustrovat si na jejich systémech a modelech významy jednotlivých důležitých pojmů.

2.1. Příklad „Písek"

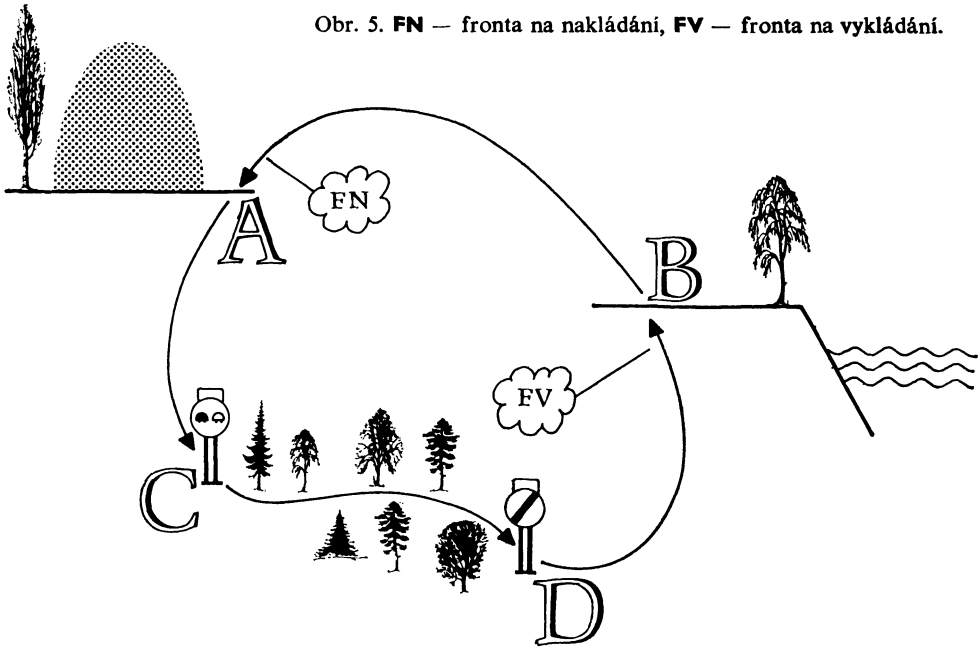
V místě „**A**" je hromada písku (mohl by to být ovšem též jiný substrát jako uhlí, zrno brambory a podobně) o známé velikosti, kterou je potřeba převézt pomocí několika nákladních aut do místa „**B**".

Každé z použitých nákladních aut vždy nejprve naloží písek v místě „**A**", odveze jej do místa „**B**" (jede přes místa „**C**" a „**D**"), kde písek vyloží a vrací se prázdné do místa „**A**" (po jiné silnici, než jelo do místa „**B**"). Je-li na hromadě v místě „**A**" ještě nějaký písek, znovu dané auto naloží a celý postup se opakuje. Po cestě jedou auta pokud možno svou maximální rychlostí a mohou se libovolně předjíždět — vyjma úseku mezi místy „**C**" a „**D**", kde je předjíždění zakázáno. (Jiná auta — kromě uvažovaných aut s pískem — zde nejezdí.)

Z technických důvodů může v místě „**A**" nakládat v každém okamžiku nejvýš jedno auto a podobně v místě „**B**" vykládat rovněž jedno auto. Ostatní auta musí čekat nenaložená nebo nevyložená, až bude v místě nakládky či vykládky volno.

Popsaný systém je schematicky znázorněn na obrázku 5.

Známe všechny potřebné silniční vzdálenosti a u každého auta známe jeho nosnost, maximální rychlost, dobu nakládky a dobu vykládky.



Obr. 5. **FN** — fronta na nakládání, **FV** — fronta na vykládání.

Úloha:

U popsaného systému je celkem přirozená otázka, za jak dlouho bude všechen písek odvezen z místa „**A**" do místa „**B**", uvažujeme-li předem danou neměnnou skupinu aut, která jsou k dispozici.

Může nás však zajímat např. i otázka, jak bude ovlivněn provoz, použijeme-li auta větší a pomalejší, resp. obráceně menší a rychlejší. Dalším možným problémem je vliv změn organizace systému, jako například dvě místa na vykládku či nakládku, přednost těžších a pomalejších aut ve frontách a podobně.

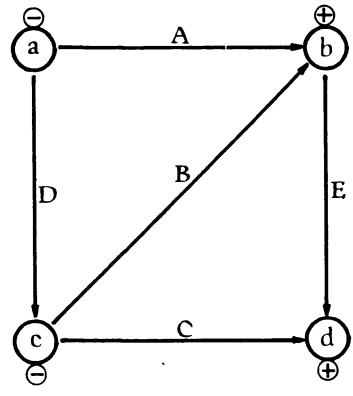
2.2. Příklad „Fotosyntéza" [15]

Biochemická struktura, na níž probíhají fotosyntetické reakce elektronového transportu, sestává z řady makromolekulárních elementů, z nichž každý může být obsazen nejvýše jedním transportovaným elektronem a mezi nimiž dochází k přeskokům elektronů z elementu na element po neměnných přechodových cestách.

K přeskoku elektronu po přechodové cestě může dojít pouze v tom případě, kdy na jejím počátečním elementu transportovaný elektron je a na koncovém nikoli. Vznikne-li pro nějakou přechodovou cestu taková situace, neprovede se přeskok elektronu okamžitě, ale až po jisté čekací době. Tyto doby čekání jsou případ od případu různé, ale pro každou přechodovou cestu jsou známa jejich statistická rozdělení.

Základním principem, kterým se elektronový transport řídí, je zákon kompetitivní inhibice. Uvažujme pro objasnění čtyři transportní elementy **a**, **b**, **c**, **d** a pět přechodo-

vých cest mezi nimi **A**, **B**, **C**, **D**, **E** podle obrázku 6. Předpokládejme, že v jistém časovém okamžiku jsou elementy **a**, **c** obsazeny elektronem, a elementy **b**, **d** nikoli. To znamená, že v takovém okamžiku jsou možné přeskoky po cestách **A**, **B**, **C**. Po cestách **D** a **E** přeskok možný není. Provede-li se ale přeskok elektronu po cestě **B**, znemožní se přeskok po cestě **A**, protože na elementu **b** bude elektron; podobně se znemožní přeskok po cestě **C**, protože na elementu **c** se již elektron vyskytovat nebude. (Konfliktní situaci, kdy by se měl současně provést přeskok třebas po cestách **A** i **B**, je možno zanedbávat.) Současně se provedením přeskoku po cestě **B** umožní přeskoky po cestách **D** a **E**. Cesta **D** již nemá obsazen koncový element a cesta **E** má nově obsazen počáteční element.



Obr. 6.

Úloha.

Jednotlivé transportní elementy je možno klasifikovat podle jejich biochemické podstaty a podle jejich polohy v celé transportní struktuře. Laboratorními metodami je možno měřit pro určité takové skupiny transportních elementů jejich saturaci elektrony, to jest zjišťovat průběh funkcí

$$\mathcal{O}_s : t \to \mathscr{E}_s^-(t)/\mathscr{E}_s ,$$

kde $\mathscr{E}_s^-(t)$ je počet elementů skupiny „**s**" obsazených v čase t elektronem a $\mathscr{E}_s$ je počet všech elementů skupiny „**s**".

Přímý tvar transportní struktury (propojení jednotlivých elementů přechodovými cestami) však laboratorně přímo zkoumat nelze. To nás přivádí k simulační úloze zjistit pomocí počítačového modelu tvar křivek funkcí $\mathcal{O}_s$ pro některé hypotetické tvary transportní struktury a ty porovnávat s přímými laboratorními výsledky.

2.3. Příklad „Křižovatka" [20]

Sledovaným systémem je provoz na křižovatce tvaru podle obrázku 7. Provoz na křižovatce je řízen světelnými signály (obvyklým způsobem červená — oranžová — zelená).

Příjezdy vozidel ve všech směrech považujeme za náhodné, popsané nějakým rozdělením pravděpodobnosti, jehož tvar není momentálně důležitý.

Při příjezdu k signálu „červená" nebo „oranžová" vozidla brzdí s konstantním zpožděním a řadí se do fronty v pořadí příjezdů; při signálu zelená pokračují přes křižovatku bez odbočování původní (maximálně povolenou) rychlostí.

Při rozjezdu z fronty na signál „zelená" se vozidla pohybují nerovnoměrně zrychleným pohybem se zrychlením zmenšujícím se s okamžitou rychlostí. Neboli exaktněji: označíme-li maximální povolenou rychlost symbolem v_M, je zrychlení a_p vozidla p při rozjíždění dáno vztahem $a_p = k_p(v_M - v_p)$, kde v_p je okamžitá rychlost vozidla p a k_p parametr jeho akcelerace. Pohyb vozidla p při rozjíždění lze tedy popsat diferenciální rovnicí:

$$dz_p^2/dt^2 = k_p(v_M - dz_p/dt)$$

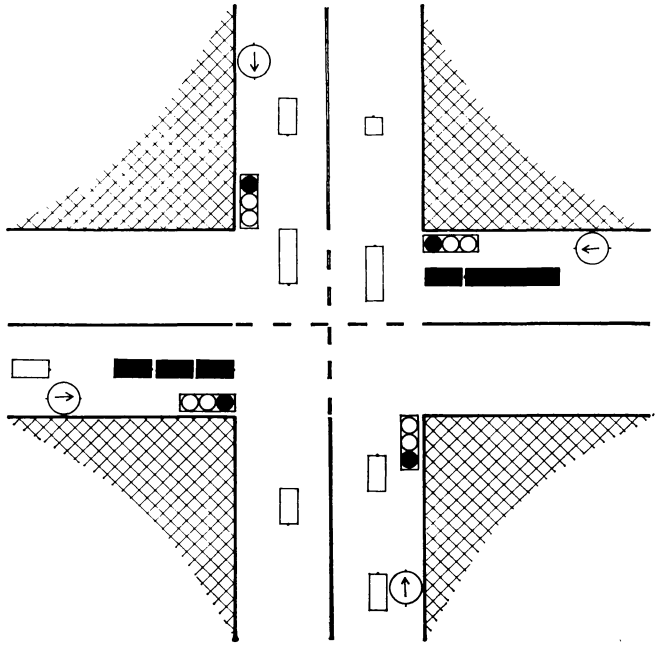
(za příslušných počátečních podmínek), kde z_p je poloha vozidla p na některé z os křižovatky.

Pohyb vozidla sledujeme v jisté zóně před křižovatkou a dále až do okamžiku, kdy opustí křižovatku na její hranici.

Úloha:

Účelem modelu křižovatky může být například zjišťování její propustnosti, která zřejmě závisí na mnoha faktorech, např. na frekvenci příjezdů vozidel, na poměru dob trvání jednotlivých světelných signálů, na maximální povolené rychlosti vozidel, na jejich počátečním zrychlení a tak podobně.

Úlohu je možno zadat i obráceně: nalézt takovou souhru parametrů provozu a řízení křižovatky, aby byla zajištěna zadaná propustnost.



Obr. 7.

Všechny systémy uvedené v příkladech se vyznačují velmi důležitou vlastností — snadno lze popsat jejich činnost v jejich jednotlivých elementárních částech a vymezit vztahy mezi těmito částmi. To je okolnost velmi příznivá pro konstrukci modelu.

Na druhé straně je chování celého systému přes zmíněnou možnost snadné dekompozice natolik složité (zejména pro veliké množství interakcí mezi jednotlivými částmi), že analytické řešení zůstává velmi obtížné nebo je dokonce prakticky nemožné.

Platí to i v případě na první pohled velmi jednoduchého systému z příkladu „Písek", jehož zadání by snad mohlo připomínat slovní úlohy z matematiky pro první stupeň základní školy. Chování jednotlivých částí systému (například jednotlivých aut) je skutečně triviální. Na druhé straně je chování celého systému již natolik složité, že i řešení nejjednodušší úlohy (za jak dlouho bude hromada známé velikosti odvezena auty známých parametrů) vyžaduje samočinný počítač. Chování celého systému může být totiž i dosti anomální — například je možné, aby se přidáním dalšího auta (a to i rychlejšího než všechna ostatní) odvezení celé hromady zpomalilo.

3. VYTVÁŘENÍ SIMULAČNÍHO MODELU

Jak plyne i ze zjednodušeného teoretického rozboru, který byl uveden výše, musí simulační program realizující model M realizovat jeho přechodovou, resp. výstupní funkci δ_m, popř. λ_m.

Konstruujeme-li simulační program na základě analýzy přechodové funkce vzorového systému $\mathcal{S}$ — což je (jak bylo též výše uvedeno) běžné, měly by stavové trajektorie výpočtu takového programu nějakým způsobem odpovídat stavovým trajektoriím originálu. (V podstatě je takový program konstruován s ohledem na nějaké mezistavové zobrazení $\psi : \mathbf{Q}_s \to \mathbf{Q}_m$.)

Hlavním úkolem programátora tedy je:

— *navrhnout strukturu dat reprezentující stavy originálního systému,*

— *navrhnout operátory nad touto strukturou, které reprezentují změny stavů,*

— *realizovat čas modelu* (*to jest realizovat časovou množinu modelu a realizovat průběh času modelu touto množinou*),

— *zajistit synchronizaci stavových změn* (*to znamená, že změny stavů se musí odehrávat v určitém pořadí a při odpovídajících hodnotách modelového času*).

3.1. Způsoby zobrazení stavů v programu

Pro reprezentaci stavů se používají nejrůznější struktury dat v závislosti na typu použitého programovacího jazyka.

V nejjednodušším případě, jsou-li stavové veličiny skalární, používají se jednoduché proměnné; podobně stavovým vektorům mohou v programu odpovídat pole dat.

V případě „Fotosyntéza" je stav popsaného systému jednoznačně dán obsazením jednotlivých elementů transportovanými elektrony. Takový stav je možno jednoduše realizovat tak, že každému transportnímu elementu bude příslušet logická proměnná, jejíž hodnoty budou při výpočtu interpretovány jako „obsazeno" — „neobsazeno".

Problémy se zobrazováním stavů v programu začínají tehdy, jestliže se struktura

systému během jeho vývoje mění nebo není konstantní počet jeho složek. Zde jsou typickými příklady otevřené systémy hromadné obsluhy s přicházejícími a odcházejícími požadavky. (Ještě se o těchto systémech zmíníme.) Při modelování systémů tohoto typu je vhodné užití programovacích jazyků, v nichž je možné přidělovat paměť exemplářům datových struktur dynamicky až v průběhu výpočtu.

V programovacích jazycích používaných pro simulaci je použití složitějších datových struktur vůbec velmi běžné. Umožňuje především popis modelovaného systému (přesněji řečeno popis jednotlivých stavů modelovaného systému) rozkládat na menší logicky vázané celky.

Tak opět v příkladě „Fotosyntéza" je pro programovou realizaci stavů systému třeba nejprve vyřešit realizaci neměnných složek systému, to jest transportních elementů a přechodových cest. Poměrně jednoduše to lze provést například realizací transportních elementů exempláři datové struktury, která obsahuje dva spojové seznamy přechodových cest (cest vedoucích do příslušného elementu a z něho).

Vlastní proměnlivý stav systému je pak možno přirozeně realizovat jednou logickou složkou v každém exempláři datové struktury odpovídajícím jednomu transportnímu elementu.

Mezi různými datovými strukturami se velmi často používají spojové seznamy, které v systémech hromadné obsluhy reprezentují fronty. V našich příkladech budou v příkladu „Písek" např. realizována pomocí spojových seznamů fronty aut čekajících na nakládání a vykládání. Každý prvek takového seznamu pak bude datovou strukturou o několika reálných složkách, které budou reprezentovat sledované vlastnosti auta — nosnost, množství momentálně naloženého písku, maximální rychlost, dobu nakládání, dobu vykládání.

Podobná situace je v příkladu „Křižovatka". Pro reprezentaci stavu vozidla stačí triviální struktura uchovávající okamžitou polohu (souřadnici) vozidla a jeho okamžitou rychlost, avšak je výhodné mít možnost tyto jednotlivé struktury vázat do spojových seznamů reprezentujících fronty u signálu „červená".

3.2. Způsoby zobrazení změn stavů v programu

Obecně vzato je každá změna stavu realizována nějakým operátorem nad stavovým prostorem programu. Tento operátor je realizován posloupností instrukcí, která má podle použitého programovacího jazyka různé formy. Systematicky vzato, může být takovou formou jak skupina příkazů, tak procedura či podprogram, ale v modernějších programovacích jazycích určených pro simulaci se většinou prostředky realizace změn stavů logicky spojují se stavovými veličinami.

V příkladu „Fotosyntéza", kde je možný jediný druh změny stavu — totiž změna stavu přeskokem elektronu po některé přechodové cestě, je možno tuto změnu realizovat procedurou (s parametrem typu cesta — přesněji typu datové struktury, kterou jsou realizovány přechodové cesty), která změní příslušnou složku ve strukturách odpovídajících začátečnímu a koncovému elementu cesty, poníž k přeskoku dochází. Avšak logičtější je nebudovat tuto proceduru samostatně, ale svázat ji s datovou strukturou pro realizaci přechodových cest.

Podobně v příkladu „Písek" lze změny stavů jednotlivých aut (realizovaných nějakou vhodnou datovou strukturou, v níž jsou zachyceny stavy aut) realizovat souborem procedur. (Např. procedura ***ZACNI_VYKLADAT*** převede auto, na které je aplikována za stavu „čekání na vykládku" do stavu „vykládá".) Výhodnější je opět programovat tyto procedury jako procedurální složky datové struktury auta.

Poněkud komplikovanější je realizace změn stavů v příkladu „Křižovatka". Stav systému je zde dán stavy jednotlivých vozidel — to jest jejich souřadnicemi na osách křižovatky — a jejich rychlostmi.

Základní procedury ***BRZDI*** (čekání na signál „červená") a ***ROZJED_SE*** (signál „zelená") jsou na rozdíl od předcházejících dvou příkladů netriviální — výpočet změny stavu vozidla se zde děje například pomocí nějaké numerické metody řešení diferenciální rovnice, která pohyb vozidla popisuje.

3.3. Zobrazení času

Zatímco dosavadní úvahy se v principu příliš nelišily od úloh obvyklého nesimulačního programování, zobrazování času a rozvíjení modelovaných činností v návaznosti na změny hodnot modelového času jsou specifickými problémy pro programování simulačních úloh.

Připomeňme motivaci času v reálných systémech, kde je chápán jako zvětšující se parametr, podle něhož jsou uspořádány všechny změny stavů systému.

Tomu musí odpovídat zobrazení času v simulačních modelech, kde je čas reprezentován aritmetickou veličinou (to jest veličinou typu **real** nebo **integer**). Shodně s reálnými systémy musí být v simulačním modelu splněna podmínka, že modelový (nazývaný též simulární) čas nesmí v průběhu výpočtu klesat.

Proto proměnná reprezentující čas není zpravidla uživateli přímo přístupná, aby ji nemohl nežádoucím způsobem modifikovat — to jest snížit hodnotu času nebo naopak zvýšit hodnotu simulárního času bez provedení odpovídajících stavových změn. Simulární čas se proto obvykle implementuje tak, že je přístupný prostřednictvím procedury, která vrací při vyvolání jeho hodnotu, ale nedovoluje přímo do časové veličiny dosazovat.

Toto chránění času před uživatelem ovšem nemůže být uplatňováno i u základních řídicích struktur programu, které synchronizují celý simulační výpočet (budeme o nich ještě podrobně hovořit). Ty mají samozřejmě k časové veličině přímý přístup — před uživatelem jsou však skryty celé.

Na rozdíl od reálných vzorových systémů není čas v simulačních modelech (jak dále bude patrné) nezávislou veličinou, na základě jejíchž hodnot by se měnily stavy systému, ale naopak je jeho hodnota zvyšována současně s prováděním předem synchronizovaných změn hodnot stavů.

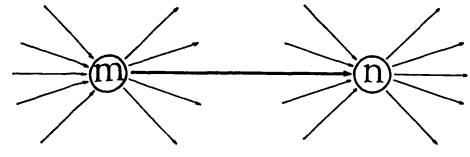
3.4. Události a jejich vztahy

Jak bylo právě uvedeno, nejsou změny stavů prováděny nahodile, ale je třeba je předem

synchronizovat. Tato potřeba synchronizace je podmíněna logickými důsledky změny stavů, které vyplývají z podstaty modelovaného systému.

Konkrétně v případě „Fotosyntéza" mohou být logické důsledky provedení přeskoku elektronu po nějaké cestě (to znamená důsledky změny stavu systému) v zásadě dvojí. Jednak je možné, že se takovým přeskokem znemožní přeskok po jiných přechodových cestách, na nichž byl dosud přeskok elektronu možný, a jednak je naopak možné, že se jiné cesty dostanou nově do stavu, kdy na nich přeskok možný je.

Přesněji, provádíme-li přeskok po přechodové cestě mezi elementy **m** a **n**, znemožníme provedení přeskoků po všech jiných cestách vedoucích z elementu **m** (na elementu **m** již transportovatelný elektron není) a po všech jiných cestách vedoucích do elementu **n** (na elementu **n** již elektron je); naopak umožníme přeskok po cestách vedoucích do elementu **m**, na jejichž začátečním elementu elektron je, a po všech cestách vedoucích z elementu **n**, na jejichž koncových elementech elektron není. (Obr. 8)



Obr. 8.

Podobně v příkladu „Písek" je mnoho logických vazeb podmiňujících logické důsledky změn stavů.

Ukončí-li některé auto v místě „**B**" vykládání, je ze stavu „vykládá" převedeno do stavu „jízda mezi místy „**B**" a „**A**"". To znamená, že dojde k uvolnění místa na vykládání a může začít vykládat další auto, dosud první ve frontě na vykládání (samozřejmě za předpokladu, že fronta na vykládání není momentálně prázdná).

To znamená, že po vyvolání procedury ***UKONCI_VYKLADANI*** aplikované na nějaké auto ve stavu „vykládá", musí bezprostředně následovat vyvolání procedury ***ZACNI_VYKLADAT*** aplikované na auto, které je prvé ve spojovém seznamu fronty na vykládání.

Podobným způsobem by bylo možné rozebírat vazby stavových změn i u systému z příkladu „Křižovatka". Zde je celkem zřejmý vztah mezi změnami světelných signálů a pohybem vozidel.

Tyto logické důsledky změn stavů určují jejich pořadí, v jakém jsou v průběhu výpočtu simulačního programu prováděny, a též určují hodnoty simulárního času, ve kterých k provádění těchto změn dochází (o určování časových hodnot bude ještě dále podrobněji pohovořeno).

Celý výpočet simulačního programu lze tedy popsat posloupností dvojic $\{t_i, \vartheta_i\}_{i=1}^{\cdots}$, kde $\{t_i\}_{i=1}^{\cdots}$ je neklesající posloupnost hodnot simulárního času a $\vartheta_i : \mathbf{Q}_m \to \mathbf{Q}_m$ jsou zobrazení nad množinou $\mathbf{Q}_m$ stavů modelu. Přesněji vzato jsou zobrazení ϑ_i v programu realizována operátory ϑ_i^* modifikujícími datovou strukturu, jejíž jednotlivé konfigurace reprezentují množinu $\mathbf{Q}_m$ stavů modelu. Operátory ϑ_i^* přitom mohou (jak záhy uvidíme) zasahovat i do řídicích struktur synchronizujících celý výpočet programu — může se tedy při vlastním výpočtu programu dít více než pouhá změna stavu $\mathbf{Q}_m \to \mathbf{Q}_m$.

Simulovaný děj je pak během výpočtu simulačního programu reprezentován posloupností akcí, z nichž každá je tvořena:

– nastavením hodnoty simulárního času na hodnotu t_i,

– provedením operátoru ϑ_i^* (a tím i změny stavu ϑ_i), pro všechna potřebná i.

Dvojici $\{t_i, \vartheta_i\}$ budeme nazývat událostí modelu, operátor ϑ_i^* událostí simulačního programu. (Použijeme-li dále jen termín událost, bude z kontextu zřejmé, kterou událost máme na mysli.)

Je nutno poznamenat, že uvedená posloupnost událostí modelu je „vnějším" projevem výpočtu simulačního programu (v podstatě je jeho trajektorii). Tato posloupnost událostí není samozřejmě v programu dána předem, ale je postupně vytvářena během výpočtu.

Pro vytváření trajektorie událostí slouží především prostředky tak zvaného plánování změn stavů.

Zmínili jsme se již o jednom důsledku procedury ***UKONCI_VYKLADANI*** v příkladě „Písek". Tato změna stavu má i jiný důsledek. Totiž auto, které přešlo do stavu „jízda mezi místy „**B**" a „**A**" ", setrvá v tomto stavu pouze po dobu potřebnou k dané jízdě (která je snadno zjistitelná ze známé vzdálenosti mezi místy „**B**" a „**A**" a známé rychlosti uvažovaného auta).

Znamená to, že po vyvolání procedury ***UKONCI_VYKLADANI*** musí dále následovat naplánování změny stavu (pro totéž auto) s převedením do stavu „čeká na nakládku" v čase, který je od momentální hodnoty simulárního času vzdálen právě o dobu jízdy uvažovaného auta mezi místy „**B**" a „**A**". Takové naplánování znamená, že plánovaná změna stavu bude provedena, až bude hodnota simulárního času rovna času plánu (nebude-li mezi tím tento plán zrušen).

Důsledkem provedení nějaké změny stavu může být též zrušení plánu události.

V naposledy uvažované situaci z příkladu „Fotosyntéza" má přeskok elektronu po přechodové cestě mezi elementy **m** a **n** tyto plánovací důsledky:

– naplánování přeskoků po všech cestách vedoucích do **m**, na nichž je přeskok možný,

– naplánování přeskoků po všech cestách vedoucích z **n**, na nichž je přeskok možný,

– zrušení plánů všech plánovaných přeskoků po jiných cestách vedoucích z **m**,

– zrušení plánů všech plánovaných přeskoků po jiných cestách vedoucích do **n**.

Pomocí plánovaní a rušení plánů změn stavů je v programu vytvářena stavová trajektorie jeho výpočtu.

O realizaci plánování a rušení plánů se podrobněji zmíníme dále.

3.5. Procesy

Výpočet simulačního programu je tedy posloupností provádění jednotlivých událostí, přičemž každá událost programu může být spojena s plánovacími akcemi. Důležité při tom je, že při provádění vlastní výkonné části události nedochází ke změně hodnoty simulárního času (každá událost se provádí celá v jednom modelovém časovém okamži-

ku) – to odpovídá výše naznačenému pojetí, že událost reprezentuje nejelementárnější změnu modelovaného systému.

Pro vlastní programování simulačních programů má však toto pojetí jednu velmi závažnou nevýhodu – totiž ve skutečných simulačních aplikacích je třeba rozlišovat velice mnoho různých typů událostí. Vlastní program, ve kterém jsou samostatně realizovány jednotlivé události, je pak velmi nepřehledný (a z toho samozřejmě plyne, že i obtížně odladitelný); nejen to – velmi nepřehledné a ve výpočtu pomalé mohou být i řídící struktury programu, které realizují plány událostí.

Tak v našem příkladě „Písek" (ač je modelovaný systém nesmírně jednoduchý a se systémy simulovanými ve skutečných aplikacích naprosto nesrovnatelný) je třeba (podle pojetí modelu) uvažovat nejméně 10 různých typů událostí.

3.5.1. Struktura procesu

Tento problém je možno odstranit zavedením tak zvaných simulačních procesů, Využijeme při tom logických návazností jednotlivých událostí a celý jejich sled v pořadí. v jakém za sebou logicky následují, sloučíme v jeden celek.

Simulační proces je tedy sled na sebe logicky navazujících simulačních událostí.

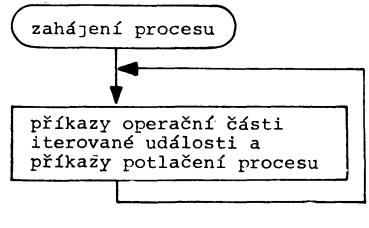
Ukázkou logické návaznosti v příkladě „Písek" může být např. skutečnost, že pro každé jednotlivé auto vždy za sebou (samozřejmě eventuálně odděleny událostmi jiných aut) následují události: ... „čekej ve frontě na nakládání" – „nakládej" – „ukonči nakládku a odjeď z fronty na nakládání" ... .

Nejpodstatnější rozdíl mezi procesem a událostí záleží v tom, že proces není prováděn celý najednou v jeden časový okamžik simulárního času – v jednom okamžiku simulárního času je realizována pouze jeho část odpovídající jedné události. Tyto části jsou od sebe odděleny tak zvanými příkazy potlačení procesu (podrobněji dále), které způsobí, že se přestane provádět daný proces a začne se uskutečňovat proces jiný – proces obsahující událost, která je nyní na řadě z hlediska celého systému.

Při dalším vyvolání uvažovaného procesu se tento proces nezačíná provádět od začátku, ale od místa, kde byl naposledy potlačen.

V některých případech nemá procesorová realizace simulačního programu příliš mnoho výhod před realizací v samostatných událostech. Je to v takových případech, kdy nejsou pevné logické vazby mezi událostmi různých typů. Například v našem příkladě „Fotosyntéza" je možno uvažovat jen jediný podstatný typ události (nepočítáme-li události při zahajování a ukončování simulace) — událost přeskoku elektronu po přechodové cestě.

Procesové pojetí simulačního modelu není však ani v těchto případech složitější než realizace v jednotlivých událostech — každé události odpovídá jeden proces, který ji iteruje — obr. 9.

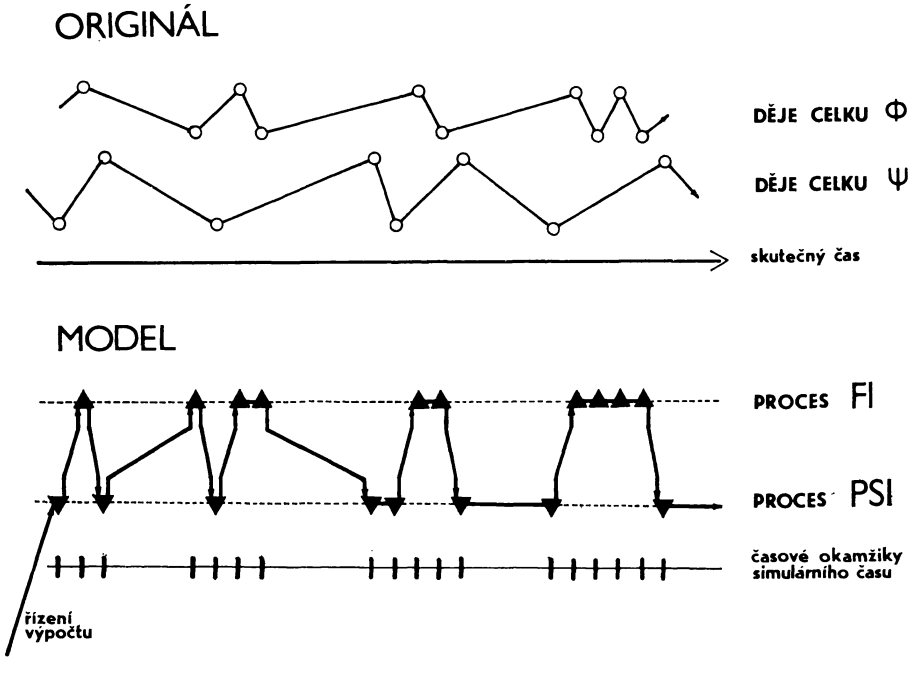


Obr. 9.

3.5.2. Plánování procesů

Podobně jako jsme výše hovořili o plánování událostí, můžeme hovořit též o plánování procesů. Význam je již z uvedeného celkem zřejmý: Je-li nějaký proces naplánován k realizaci v jistém časovém okamžiku, provede se ve fázi výpočtu, kdy je hodnota simulárního času rovna času uvažovaného plánu, část daného procesu od jeho posledního přerušení k následujícímu.

Z prakticky programátorského hlediska slouží potlačování procesů a s ním spojené předávání řízení výpočtu mezi jednotlivými procesy k tomu, aby bylo možno modelovat simultánně probíhající děje v originálním systému sekvenčně prováděnými operacemi monoprocesorového počítače – obr. 10. Tak v příkladě „Křižovatka" probíhají děje dvou vozidel přijíždějících z různých stran ve skutečnosti současně, ale v sekvenčně prováděném modelu se musí provádění obrazů těchto dějů střídat.



Obr. 10.

Poznamenejme ještě, že z logické následnosti událostí v procesu plyne, že každý proces (přesněji každý exemplář procesu utvořený v programu) může mít v každém okamžiku výpočtu nejvýše jeden plán. (Pro události je analogická poznámka evidentní.)

3.5.3. Vnější stavy procesů

V souvislosti s plánováním procesů rozlišujeme čtyři různé stavy procesů. Každý exemplář procesu vytvořený v průběhu výpočtu simulačního programu se v každém okamžiku výpočtu vyskytuje v právě jednom z těchto stavů:

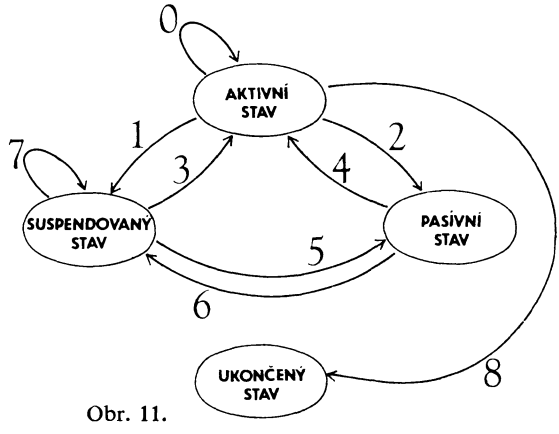
– *Stav aktivní.* Proces je ve stavu aktivním, je-li právě prováděn (v každém okamžiku výpočtu je zřejmě takový proces nejvýše jeden).

– *Stav ukončený.* Naopak, proces je ve stavu ukončeném, je-li již ukončena jeho operační část, což znamená, že již nemůže být vyvolán ani naplánován.

– *Stav suspendovaný.* V tomto stavu je proces, který není aktivní, ale je naplánován k provedení v nějakém okamžiku modelového času.

– *Stav pasívní.* Konečně v tomto stavu je proces, který není ukončen, ale není momentálně naplánován k provedení.

Příkazy pro vytváření, resp. rušení plánů procesů je pak možno klasifikovat podle toho, jak který příkaz mění vnější stav procesu, ke kterému se vztahuje. Z logiky věci plyne, že je možno v tomto smyslu uvažovat osm různých typů efektů synchronizačních příkazů (viz obr. 11.) – přechod do ukončeného stavu se děje automaticky vyčerpáním operační části procesu. Přechody ze stavu ukončeného jinam jsou principiálně nemožné.



Obr. 11.

3.5.4. Klasifikace změn vnějších stavů

Některé z těchto plánovacích příkazů jsou právě ty, které způsobují potlačení momentálně prováděného procesu. Právě z tohoto hlediska si rozebereme jednotlivé možnosti plánovacích příkazů (podle schématu na obr. 11.).

1. Změna stavu *aktivní* → *suspendovaný*

Nejčastěji k této změně dojde, je-li zrušen plán dosud aktivního procesu, podle něhož je právě prováděn, a další provádění tohoto procesu je naplánováno až po jisté době simulárního času. Tak v případě „Písek" je např. proces auta, které právě ukončilo vykládku a odjíždí z fronty na vykládání, suspendován po dobu jízdy tohoto auta mezi místy „**B**" a „**A**" – po uplynutí této doby dojede auto do místa „**A**", čemuž budou odpovídat další příkazy uvedené v procesu auta po tomto potlačení.

V popsaném případě je možné, že nějaký jiný proces je naplánován s časem menším,

než je čas nového plánu dosud aktivního procesu, a v takovém případě se aktivním stane proces s minimálním časem plánu ze všech takových (kolize při stejných časech plánů více procesů se řeší lineárním uspořádáním takové skupiny plánů, fixními prioritami procesů či podobně).

Může se ovšem též stát, že žádný jiný proces není naplánován s časem menším, než je čas nového plánu dosud aktivního procesu. Pak dojde pouze ke změně hodnoty simulárního času a dosud aktivní proces bude v aktivním stavu i nadále — simulární čas bude roven času jeho nového plánu. (Lze tedy vlastně uvažovat i „změnu" stavu *aktivní* → *aktivní* — změna 0 schématu na obr. 11.)

Ke změně stavu aktivního do suspendovaného může dojít též vedlejším efektem při přímé aktivaci jiného procesu (viz dále body 3. a 4.).

2. Změna stavu *aktivní* → *pasívní*

V tomto případě dojde ke zrušení momentálního plánu dosud aktivního procesu, ale jeho další pokračování není dále explicitně plánováno. Dosud aktivní proces setrvá pak v pasívním stavu do té doby, dokud jej nějaký jiný proces nepřevede do stavu suspendovaného nebo přímo aktivního.

V našem příkladě „Písek" se do pasívního stavu převede proces auta, které dojelo do místa „**A**" a zařadilo se do fronty na nakládání. Zde bude čekat v pasívním stavu (samozřejmě nebyla-li dosud fronta na nakládání prázdná), dokud na odpovídající auto nedojde ve frontě na nakládání řada, aby začalo nakládat (příkazy procesu následující za potlačením). Opětovné spuštění aut čekajících ve frontě je možno realizovat tak, že proces auta, které odjíždí z místa „**A**" do místa „**C**" po ukončení nakládky, před tím, než přejde do suspendovaného stavu, převede svého následníka ve frontě na nakládání (je-li jaký) ze stavu pasívního do stavu aktivního (viz. dále bod 4.).

Podobně jako v předcházejícím případě se po pasivaci dosud aktivního procesu převede do aktivního stavu ten ze suspendovaných procesů, který má nejmenší čas plánu. (Poznamenejme v této souvislosti, že simulační systémy pracující na procesovém principu obvykle vyžadují, aby byl alespoň jeden proces ve stavu aktivním nebo suspendovaném — převedení všech procesů (neukončených) do stavu pasívního buď ukončí výpočet simulačního programu, nebo vede přímo k chybě při výpočtu.)

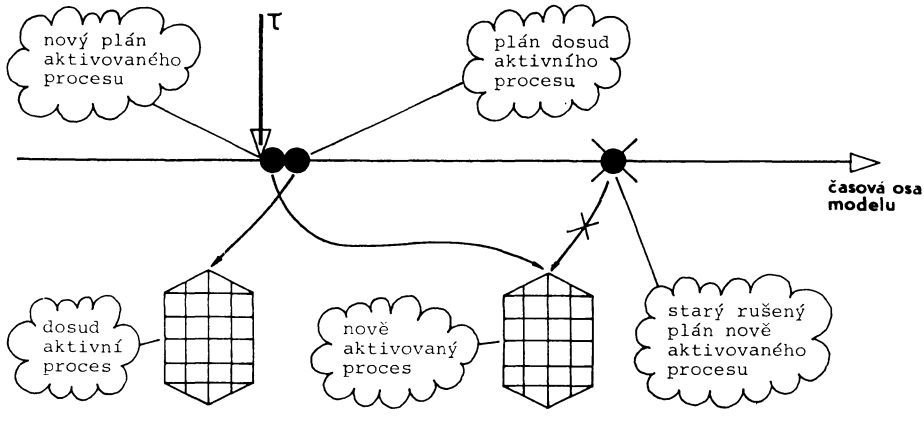
3. Změna stavu *suspendovaný* → *aktivní*

Ke změně vnějšího stavu procesu ze suspendovaného do aktivního může dojít (jak jsme se již výše zmínili) vedlejším efektem při potlačování jiného procesu.

Může však dojít k takové změně i přímo a znamená to, že zrušíme existující plán suspendovaného procesu a naplánujeme jej s časem rovným momentální hodnotě simulárního času a začneme jej provádět. Tak v případě „Křižovatka" může být např. proces vozidla přijíždějícího k hranici křižovatky na červenou suspendován po dobu brzdění. Dojde-li však ke změně světelného signálu dřív, než vozidlo zastaví, je třeba okamžitě předat řízení výpočtu procesu brzdícího vozidla, aby se toto vozidlo opět dostalo do rozjížděcí fáze.

Nově aktivní proces má pak nový plán provádění s časem stejným, jako je plán provádění dosud aktivního procesu (který jsme nezrušili) — viz obr. 12. Až bude nově aktivní proces potlačen, dojde opět k vyvolání dříve aktivního procesu (nedojde-li mezitím k dalším změnám plánů procesů).

Přesněji: Proces, který byl dosud aktivní, zůstává naplánován s časem rovným momentální hodnotě simulárního času, ale jeho plán je „až druhý na řadě" — „před ním" je plán nově zaktivovaného procesu. Později — až dojde k potlačení nově aktivního procesu, bude opět plán původně aktivního, procesu „první na řadě" (nebude-li mezitím změněn provedenými akcemi nově aktivního procesu) a opět bude tento proces převeden do aktivního stavu. Avšak ke změně hodnoty simulárního času přitom nedojde.



Obr. 12. τ — momentální hodnota simulárního času.

4. Změna stavu *pasívní → aktivní*

Předchozímu případu je do jisté míry podobná změna stavu z pasívního do aktivního. Proces, který nebyl momentálně naplánován, naplánujeme k okamžitému provedení, čímž převedeme dosud aktivní proces do suspendovaného stavu (dojde k jeho potlačení).

O použití této změny vnějšího stavu procesu v realizaci modelu z příkladu „Písek" jsme se zmínili výše v bodě 2.

Všechny tři dosud probírané změny vnějších stavů procesů měly (nebo mohly mít) za následek potlačení dosud aktivního procesu. Zbývající tři možnosti změn vnějších stavů procesů potlačení aktivního procesu nepůsobí a týkají se pouze obhospodařování plánů procesů.

5. Změna stavu *suspendovaný → pasívní*

Změna stavu procesu ze suspendovaného do pasívního znamená pouze zrušení jeho dosud existujícího plánu. Tak uvažujeme-li situaci z příkladu „Fotosyntéza" podle obrázku 8, pak při provedení přeskoku elektronu z elementu **m** na element **n** musí být převedeny do pasívního stavu všechny procesy iterující události přeskoků po jiných cestách z elementu **m** a jiných cestách do elementu **n**, které jsou v suspendovaném stavu (tj. naplánovány k provedení na nějakou pozdější dobu).

6. Změna stavu *pasívní → suspendovaný*

Naopak změna stavu procesu z pasívního do suspendovaného znamená naplánování aktivace dosud nenaplánovaného procesu (avšak nikoli k okamžité aktivaci — k okamžitému provedení).

V právě uvažované situaci z příkladu „Fotosyntéza" je třeba při provedení přeskoku **m** → **n** nově naplánovat dosud pasívní procesy iterující přeskoky po cestách do **m** a z **n**, po nichž byl přeskok právě umožněn. K přeskokům však nedojde ihned, ale až po jisté době, jejíž velikost můžeme třebas náhodně generovat ze známých statistických rozdělení. Iterující procesy nejsou proto plánovány k okamžité aktivaci.

7. změna plánu procesu *v suspendovaném stavu*

Poslední explicitní možností, jak ovlivňovat plány procesů, je změnit plán procesu, který byl a zůstává v suspendovaném stavu (tj. změnit čas, v němž jej plánujeme k aktivaci).

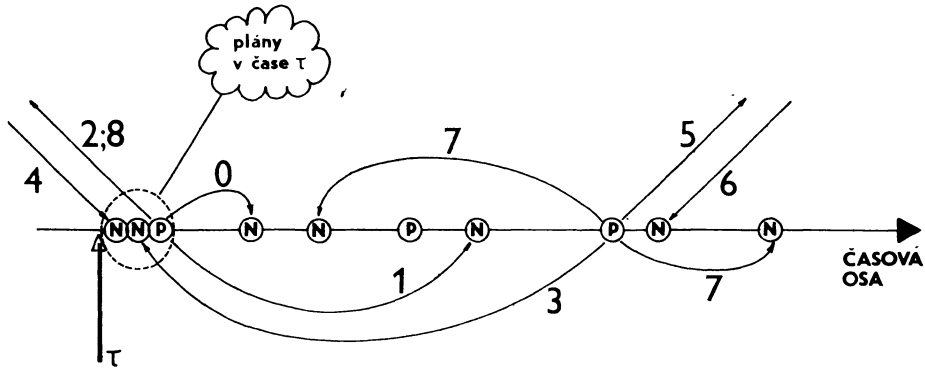
Třebas v systému z příkladu „Křižovatka" můžeme navíc uvažovat změny počasí. Pak je třeba, aby proces vozidla přijíždějícího k hranici křižovatky na červenou — suspendovaný po dobu brzdění — byl v případě, že začne náhle pršet, přeplánován s pozdějším časem plánu, neboť doba brzdění vozidla se prodlouží.

Pro úplnost se zmíníme ještě o důsledcích vyčerpání operační části procesu.

8. Změna stavu *aktivní → ukončený*

Při vyčerpání operační části procesu se samozřejmě zruší též plán tohoto procesu a řízení se předá procesu s dalším časově nejmenším plánem. (Jak se ještě zmíníme později, může mít ukončení operační části některého procesu za následek ukončení celého simulačního výpočtu.)

Povšimněme si ještě, že všechny probrané možnosti změn stavů procesů musí být vyvolány (procedurou či speciálním příkazem — podle toho, jak jsou realizovány) v aktivním procesu, neboť veškerý výpočet simulačního programu se při procesové realizaci odehrává pouze v procesech. Změny typu ad 0, 1 a 2 (a samozřejmě 8) může tedy proces provést na sobě sám, změny všech ostatních typů může na procesu provést pouze proces jiný (který je momentálně aktivní a který se změnami typů ad 3 a 4 potlačí). Pro názornost obrázek 15 schematicky ještě uvádí změny plánů procesů ve všech devíti uvedených případech.



Obr. 13. **P** — původní plány procesů, **N** — nové plány procesů, τ — momentální hodnota simulárního času.

3.5.5. Vnitřní stavy procesů

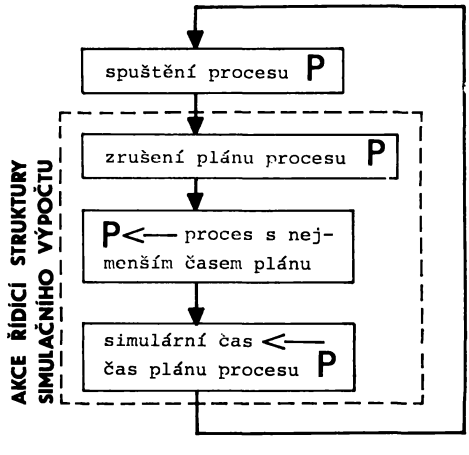
Poměrně složitý systém potlačování procesů umožňuje na druhé straně poněkud zjednodušit některé jiné složky simulačního programu. To se týká především zobrazování stavů modelovaného systému. Prvky systému (přesněji jeho stavového prostoru) jsou totiž převáděny do různých stavů právě prováděním operačních částí procesů (přesněji úseky operačních částí procesů mezi dvěma potlačeními, které odpovídají různým událostem).

Je tedy celkem přirozené zobrazovat stavy modelovaného systému pomocí tzv. vnitřních stavů procesů; tím rozumíme místo v operační části procesu, v němž je daný exemplář procesu přerušen*).

3.6. Kalendáře

Systémy plánování událostí či procesů vyžadují speciální prostředky, které plány registrují, umožňují jejich změny (nové plánování a rušení existujících plánů) a vydávají událost či proces s nejmenším časem plánu.

Takové prostředky jsou základní řídící strukturou simulačního programu — jeho výpočet probíhá totiž tak, že po ukončení události nebo po potlačení procesu se řízení výpočtu předá této řídící struktuře, která nalezne událost či proces s nejmenším časem plánu (počítaje v to různé priority při více plánech se stejným časem), na tento čas zvýší hodnotu simulárního času a spustí nalezenou událost či proces — obr. 14.



Obr. 14.

Popsanou řídící strukturu simulačního programu nazýváme simulačním kalendářem.

Pro jednoduchost budeme dále hovořit o simulačních kalendářích programů s plánováním procesů — pro případ plánování událostí platí však vše dále uvedené pouze s nepodstatnými změnami.

*) Též tzv. reaktivační bod procesu.

3.6.1. Abstraktní typ kalendáře

Abstraktně se lze na simulační kalendář dívat jako na datovou strukturu sestávající z údajové části a tří zpřístupňujících operací (plánování procesů – operace **A**, rušení plánů – operace **C** a nalézání procesu prvního na časové ose – operace **F**).

Na tyto zpřístupňující operace lze formálně pohlížet jako na zobrazení:

$$\mathscr{A} : K \times P \times \mathrm{E}_1^+ \to K, \mathscr{C} : K \times P \to K, \mathscr{F} : K \to P,$$

kde K je množina právě všech hodnot údajové části kalendáře, P množina právě všech exemplářů procesů v programu a E_1^+ je množina právě všech nezáporných reálných čísel.

Přitom hodnotu údajové části kalendáře $h \in K$ (dále jen hodnotu kalendáře) lze jednoznačně popsat zobrazením

$$h^{\S} : \mathrm{dom}\,(h^{\S}) \to \mathrm{E}_1^+, \mathrm{dom}\,(h^{\S}) \subseteq P .$$

Množinu dom $(h^{\S})$ chápeme jako množinu právě všech procesů v aktivním nebo suspendovaném stavu, proces $p \in \mathrm{dom}\,(h^{\S})$ je plánován k provedení v čase $h^{\S}(p)$.

Operace **F** na základě hodnoty kalendáře $h \in K$ vydá proces $\mathscr{F}(h) \in \mathrm{dom}\,(h^{\S})$ takový, že

$$(\forall p \in P)\,(p \in \mathrm{dom}\,(h^{\S}) \to h^{\S}(p) \geqq h^{\S}(\mathscr{F}(h)))\,.$$

Operace **A** na základě hodnoty kalendáře $h \in K$, procesu $p \in P$ a nezáporného reálného čísla c vydá hodnotu kalendáře $\mathscr{A}(h, p, c)$, pro kterou platí:

$$\mathrm{dom}\,(\mathscr{A}(h, p, c)^{\S}) = \mathrm{dom}\,(h^{\S}) \cup \{p\}\,,$$

$$\mathscr{A}(h, p, c)^{\S} \diagup\!\!\diagdown (\mathrm{dom}\,(h^{\S}) - \{p\}) = h^{\S} \diagup\!\!\diagdown (\mathrm{dom}\,(h^{\S}) - \{p\})\,,$$

$$\mathscr{A}(h, p, c)^{\S}(p) = c\,.$$

Konečně operace **C** vydá na základě hodnoty kalendáře $h \in K$ a procesu $p \in P$ hodnotu kalendáře $\mathscr{C}(h, p)$ takovou, že

$$\mathscr{C}(h, p)^{\S} = h^{\S} \diagup\!\!\diagdown (\mathrm{dom}\,(h^{\S}) - \{p\})\,.$$

3.6.2. Programová realizace

Celkem snadno lze nahlédnout, že pro programovou realizaci simulačního kalendáře vyhovuje jakákoli struktura realizující lineární uspořádání na konečné množině; univerzální simulační systémy v simulačních programovacích jazycích nebo v jazycích pro simulaci často používaných také vesměs obsahují simulační kalendáře ve formě spojového seznamu nebo binárního stromu.

Tyto struktury se též často používají v případech, kdy strukturu simulačního kalendáře vytváříme přímo v uživatelské části programu.

V případech některých simulačních modelů je výhodné (jak dále uvidíme) pro realizaci kalendáře použít strukturu tak zvaného bezkalendáře: Je-li P neměnná množina procesů v programu $P = \{p_1, p_2, \ldots, p_n\}$, pak lze údajovou část kalendáře realizovat polem reálných hodnot $H[1 : n]$ tak, že pro hodnotu h kalendáře platí:

$$p_i \in \mathrm{dom}\,(h^{\S}) \to H[i] = h^{\S}(p_i)\,,$$

$$p_i \notin \mathrm{dom}\,(h^{\S}) \to H[i] < 0\,.$$

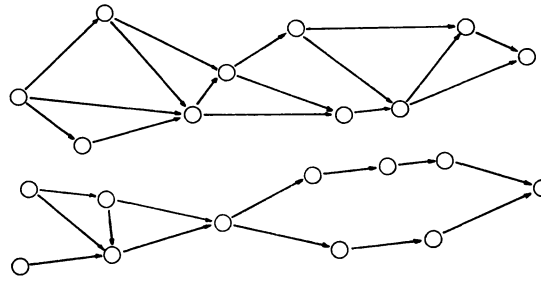
Algoritmy **A** a **C** lze realizovat velmi jednoduše pouhou změnou příslušných prvků v poli H (jsou

složitosti $O(1)$), algoritmus **F** realizujeme jednoduchým cyklem, v němž nalezneme nejmenší nezápornou hodnotu složek pole H (je složitosti $O(n)$).

3.6.3. Hierarchické kalendáře

Realizace kompletního lineárního uspořádání množiny plánovaných procesů není však na druhé straně pro konstrukci simulačního kalendáře nezbytná. (Nepotřebujeme kompletní uspořádání, potřebujeme pouze nalézat nejmenší prvek.) Lze toho využít při speciálních konstrukcích simulačních kalendářů, jejichž algoritmy pracují efektivněji než algoritmy struktur kompletního uspořádání. To se týká především simulačních modelů, jejichž procesy lze bez algoritmické náročnosti třídit do více skupin.

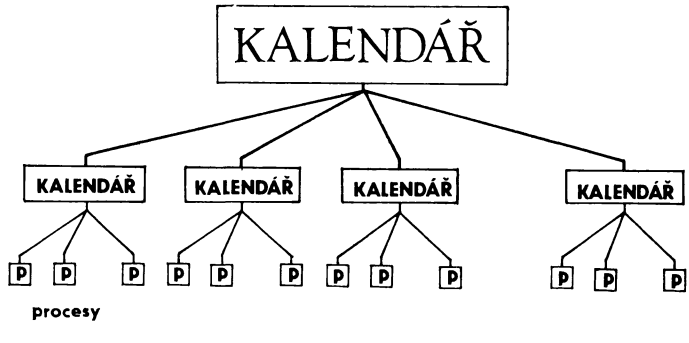
Tak simulujeme-li v příkladě „Fotosyntéza" elektronový transport po transportní struktuře složené ze dvou samostatných částí (např. podle obr. 15), lze o každém procesu realizujícím transport po některé přechodové cestě jednoznačně prohlásit, ke které části systému patří. Navíc bude mít provedení takového procesu za následek změny pouze těch plánů, které příslušejí procesům stejné části transportní struktury.



Obr. 15.

To nás přivádí k myšlence zkonstruovat pro obě dvě části modelovaného systému samostatné struktury kalendáře (o menším rozsahu, a tedy i efektivnější) a v každém kroku výpočtu porovnávat pouze hodnoty prvních plánů těchto samostatných částí.

Obecněji lze uvažovat i o větším množství samostatných struktur kalendáře, mezi nimiž opět potřebujeme hledat minimum z jejich minimálních plánů, a tedy je můžeme svázat dohromady nadřazenou strukturou dalšího simulačního kalendáře – obr.16.



Obr. 16.

Přesněji:

Nechť je množina právě všech procesů P disjunktně rozdělena na konečný počet neprázdných částí $P = \cup_{i=1}^{n} P_i$ a nechť je nad každou množinou P_i zkonstruován kalendář Ψ_i s algotitmy $\mathbf{F}_i$, $\mathbf{A}_i$, $\mathbf{C}_i$ a nad množinou $P^- = \{P_1, \ldots, P_n\}$ kalendář Ψ^- s algoritmy $\mathbf{F}^-$, $\mathbf{A}^-$, $\mathbf{C}^-$ takový, že pro každé i = 1, 2, ..., n platí:

$$h(\Psi^-)^{\S}\ (P_i) = h(\Psi_i)^{\S}\ (\mathscr{F}_i(h(\Psi_i)))\,,$$

kde $h(\Phi)$ je hodnota údajové části kalendáře Φ.

Pak můžeme nad množinou P zkonstruovat kalendář Ω s algoritmy $\mathbf{F}$, $\mathbf{A}$, $\mathbf{C}$ takový, že
$(\forall p \in P, i = 1, \ldots, n)\,(p \in P_i \rightarrow h(\Omega)^{\S}\,(p) = h(\Psi_i)^{\S}\ (p))$.

Konstrukci tohoto kalendáře provedeme takto:

Algoritmus $\mathbf{F}$ nalezne j takové, že $P_j = \mathscr{F}^-(h(\Psi^-))$ a vydá prvek $\mathscr{F}_j(h(\Psi_j))$.

Aplikujeme-li algoritmus $\mathbf{A}$ na prvek p a reálné číslo c, pak existuje P_i takové, že $p \in P_i$. Je-li $c \geqq h(\Psi_i)^{\S}\ (\mathscr{F}_i(h(\Psi_i)))$,

pak změníme hodnotu kalendáře Ψ_i na $\mathscr{A}_i(h(\Psi_i), p, c)$; v opačném případě změníme navíc hodnotu kalendáře Ψ^- na

$$\mathscr{A}^-(\mathscr{C}^-(h(\Psi^-), P_i), P_i, c)\,.$$

Podobně, aplikujeme-li algoritmus $\mathbf{C}$ na prvek p, pak existuje P_i, $p \in P_i$. Je-li $p \neq \mathscr{F}_i(h(\Psi_i))$, pak změníme hodnotu kalendáře Ψ_i na $\mathscr{C}_i(h(\Psi_i), p)$; v opačném případě změníme navíc hodnotu kalendáře Ψ^- na

$$\mathscr{A}^-(\mathscr{C}^-(h(\Psi^-), P_i), P_i, h(\Psi_i)^{\S}\ (\mathscr{F}_i(h(\Psi_i))))\,.$$

Zkonstruovaný kalendář Ω nazýváme potom hierarchickým kalendářem (nebo hierarichií kalendáře Ψ^- nad kalendáři Ψ_i).

Pro složitosti jeho algoritmů lze odvodit odhady:

$$s(\mathbf{F}) \lesssim s(\mathbf{F}^-) + \max_{i=1,\ldots,n} s(\mathbf{F}_i)\,;$$

$$s(\mathbf{A}) \lesssim \max(\max_{i=1,\ldots,n} s(\mathbf{A}_i); \max_{i=1,\ldots,n} s^{\|}(\mathbf{A}_i) + s(\mathbf{C}^-) + s(\mathbf{A}^-))\,;$$

$$s(\mathbf{C}) \lesssim \max(\max_{i=1,\ldots,n} s(\mathbf{C}_i); \max_{i=1,\ldots,n} s^{\|}(\mathbf{C}_i) + s(\mathbf{F}_i) + s(\mathbf{C}^-) + s(\mathbf{A}^-))\,;$$

kde $s(\mathbf{Q})$ je složitost algoritmu $\mathbf{Q}$, $s^{\|}(\mathbf{A}_i)$ je složitost algoritmu $\mathbf{A}_i$ při zařazování na začátek časové osy kalendáře a podobně $s^{\|}(\mathbf{C}_i)$ složitost rušení plánu s minimálním časem.

Jsou-li např. množiny $P^-, P_1, P_2, \ldots, P_n$ stejné mohutnosti a kalendáře $\Psi^-, \Psi_1, \Psi_2, \ldots, \Psi_n$ jsou zkonstruovány ve formě spojových seznamů, lze pro složitosti algoritmů hierarchického kalendáře Ω odvodit vztahy:

$$s(\mathbf{F}) \sim O(1)\,;$$

$$s(\mathbf{A}) \sim s(\mathbf{C}) \sim O(\sqrt{(\text{card } P)})\,.$$

Kalendáře Ψ_i mohou být ovšem zkonstruovány též v podobě hierarchických; tak lze dospět k hierarchiím vyšších úrovní. Např. pro vícenásobně hierarchický kalendář z binárních stromů ve tvaru tak zvaného radikálového stromu (viz. [7]) lze odvodit odhad složitostí algoritmů:

$$s(\mathbf{F}) \sim s(\mathbf{A}) \sim s(\mathbf{C}) \sim O(\log \log (\text{card } P))\,.$$

3.6.4. Hybridy

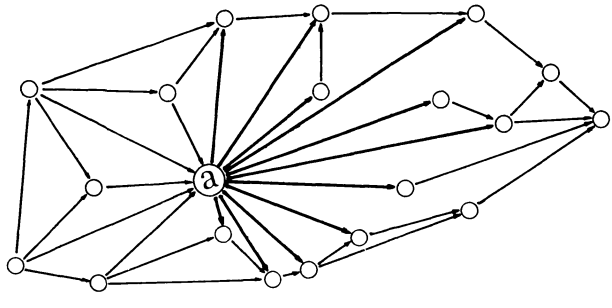
Z praktického hlediska se jeví být důležitější než různé soustavy balancovaných binárních stromů hierarchické struktury kalendáře, jejichž algortimy jsou přizpůsobeny vnitřní logice modelovaného systému.

Uvažujme v příkladě „Fotosyntéza" třídění procesů přechodových cest podle toho,

ze kterého elementu vycházejí — obr. 17. Pro každou takovou skupinu procesů lze vytvořit samostatný kalendář a z nich hierarchický kalendář pro celý modelovaný systém.

Předpokládejme dále, že pro nějaký element **a** mají přechodové cesty z něho vycházející doby čekání před realizací přeskoku nepoměrně vyšší než přechodové cesty vedoucí do a z jejich koncových elementů. Než se provede přeskok elektronu z elementu **a**, bude se mnohokrát zasahovat (nově plánovat a rušit plány) do kalendáře odpovídajícího soustavě cest vedoucích z elementu **a**. Předpokládejme, že takových zásahů bude m od okamžiku, kdy přijde elektron na element **a**, do okamžiku, kdy se realizuje přeskok z elementu **a**. Lze odvodit, že bude-li příslušný kalendář realizován binárním stromem, bude složitost operací na něm provedených v průběhu jednoho popsaného kroku výpočtu (přechod na **a** → odchod z **a**) odhadnutelná veličinou $m \cdot \log n$, kde n je počet přechodových cest vycházejících z elementu **a**. Podobně lze pro strukturu bezkalendáře (zmínili jsme ji v paragrafu 3.6.2.) odvodit složitost operací na kalendáři během tohoto kroku výpočtu $\sim m + n$. Rozdíl obou odhadů je pro $m \gg n \gg 1$ nezanedbatelný.

Máme-li tedy na zřeteli vnitřní zákonitosti modelovaného systému i při konstrukci simulačního kalendáře, je výhodné používat hierarchických konstrukcí, které ve svých jednotlivých složkách používají různé algoritmy (hybridní kalendáře).



Obr. 17.

3.7. Čekání na podmínku

Plánování událostí či procesů vazbou jejich akcí na nějakou hodnotu simulárního času lze dále zobecnit. Spuštění procesu nebo provedení události lze vázat libovolnou podmínkou, která se může týkat jak modelovaných stavů systému, tak plánů ostatních událostí či procesů i hodnot simulárního času.

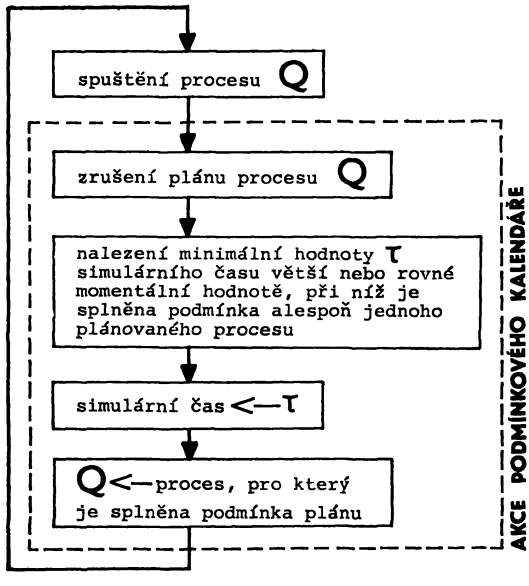
Takové plánování v podmínkách značně komplikuje základní řídící struktury simulačního programu (podmínkové kalendáře); speciálně je značně komplikované zvyšování hodnot simulárního času — obr. 18.

Po abstraktní datové stránce má podmínkový kalendář typ analogický typu normálního časového kalendáře.

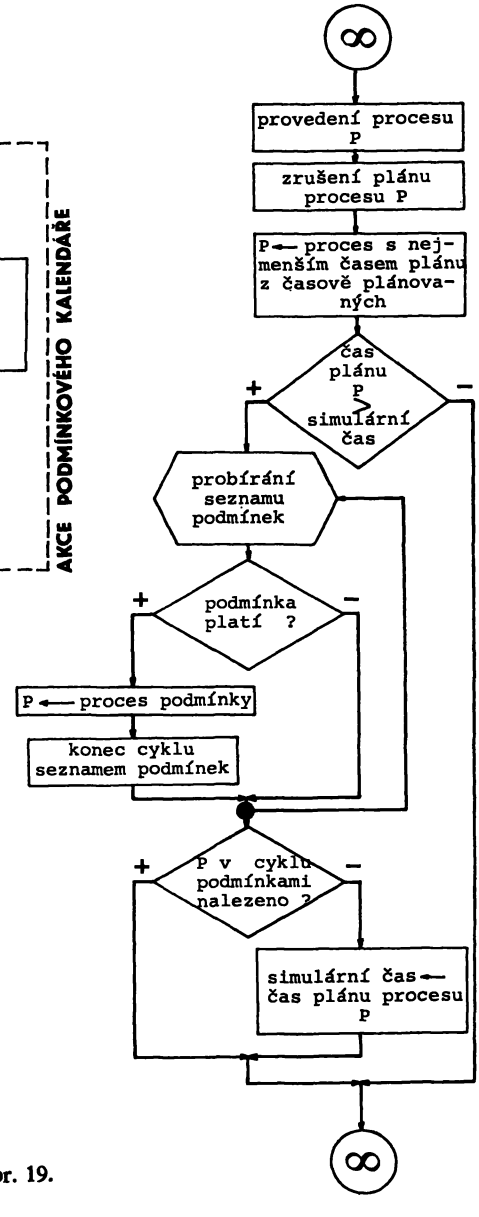
Skutečně programová realizace podmínkového kalendáře ve vší obecnosti je značně náročná.

Naštěstí není v aplikačních simulačních programech potřeba skutečného podmínko-

vého plánování častá, a je-li již třeba podmínkově plánovat, je to prakticky vždy plánování pomocí podmínek, které se týkají pouze hodnot stavového prostoru modelu (nikoli plánů ostatních procesů či hodnot simulárního času).



Obr. 18.



Obr. 19.

V takových modelech se pak vyskytují jednak procesy, které jsou plánovány podmínkami zmíněného druhu, a jednak procesy, které jsou plánovány obvyklým časovým způsobem. Vazba podmínek pouze na stavový prostor modelu znamená, že zvyšování hodnoty simulárního času se může dít stejným způsobem jako u prostého časového plánování.

Celou řídící strukturu výpočtu je možno realizovat tak, že podmínkové plány uspořádáme podle nějaké priority a po každé změně modelovaných stavů systému (tj. například před každým zvyšováním hodnoty simulárního času) zkontrolujeme po řadě podmínkové plány a spustíme první nalezený proces (podmínkově plánovaný), jehož podmínka je splněna. Zakládání nebo rušení podmínkových plánů záleží pak pouze v jejich zařazování nebo vyřazování ze seznamu podmínek.

Schéma jedné možné realizace řídící struktury je na obr. 19. (Časově plánované procesy mají v tomto schématu přednost před podmínkově plánovanými; to samozřejmě není podstatné a nejrůznější priority mezi procesy lze realizovat i jinak.) [8]

Dokončení v příštím čísle.

O Hardyho nerovnosti

Alois Kufner, Praha

1. Tzv. *Hilbertova věta o dvojných řadách* říká, že pro nezáporné posloupnosti $\{a_n\}$, $\{b_n\}$ $(n \geqq 1)$ platí

$$\sum_m \sum_n \frac{a_m b_n}{m+n} \leqq \frac{\pi}{\sin(\pi/p)} \left(\sum_n a_n^p\right)^{1/p} \left(\sum_n b_n^{p'}\right)^{1/p'}; \tag{1.1}$$

zde je $p > 1$ a $p' = p/(p-1)$. Platí i integrální analogie nerovnosti (1.1):

$$\int_0^\infty \int_0^\infty K(x,y) f(x) g(y)\,\mathrm{d}x\,\mathrm{d}y \leqq c_0 \left(\int_0^\infty f^p(x)\mathrm{d}x\right)^{1/p} \left(\int_0^\infty g^{p'}(x)\,\mathrm{d}x\right)^{1/p'}, \tag{1.2}$$

a to nejen pro jádro $K(x,y) = 1/(x+y)$, nýbrž i pro obecnější jádra. Speciálně platí pro nezáporná jádra $K(x,y)$, jež jsou homogenní stupně -1, vedle (1.2) též nerovnosti

$$\int_0^\infty \left[\int_0^\infty K(x,y) f(x)\,\mathrm{d}x\right]^p \mathrm{d}y \leqq c_0^p \int_0^\infty f^p(x)\,\mathrm{d}x, \tag{1.3}$$

$$\int_0^\infty \left[\int_0^\infty K(x,y)\, g(y)\,\mathrm{d}y\right]^{p'} \mathrm{d}x \leqq c_0^{p'} \int_0^\infty g^{p'}(y)\,\mathrm{d}y$$

s konstantou

$$c_0 = \int_0^\infty K(x,1)\, x^{-1/p}\,\mathrm{d}x = \int_0^\infty K(1,y)\, y^{-1/p'}\,\mathrm{d}y.$$