UČEBNÍ TEXTY OSTRAVSKÉ UNIVERZITY

Přírodovědecká fakulta

SIMULACE A MODELOVÁNÍ

Ivan Křivý a Evžen Kindler

OSTRAVSKÁ UNIVERZITA 2001

OSTRAVSKÁ UNIVERZITA

SIMULACE A MODELOVÁNÍ

Ivan Křivý a Evžen Kindler

Předmluva

Tato skripta jsou určena především studentům magisterských studijních programů „Aplikovaná matematika“ a „Informatika – informační systémy“ jako učební text ke stejnojmennému kurzu; mohou však dobře posloužit i studentům jiných programů, resp. oborů, magisterského studia na Přírodovědecké fakultě Ostravské univerzity. Autoři předpokládají u čtenářů pouze základní znalosti z matematické analýzy a programování, získané v průběhu prvních dvou let studia.

Skripta seznamují čtenáře se základními pojmy v oblasti modelování a simulace, principy algoritmizace simulačních modelů diskrétních i spojitých systémů a programovacími prostředky pro modelování a simulaci. Obsahují také přehled matematických prostředků a metod (teorií) běžně používaných v praxi modelování a simulace. Samostatná kapitola je věnována příkladům na simulaci, které zahrnují systémy hromadné obsluhy, různé přístupy k evoluci buněčných systémů (zejména znalostní přístup a přístup založený na teorii Lindenmayerových systémů) a simulaci šíření epidemie (aplikace teorie celulárních automatů a modelu větvícího se procesu).

Autoři děkují touto cestou recenzentu RNDr. Jiřímu Weinbergerovi za pečlivé pročtení rukopisu a řadu cenných připomínek směřujících ke zkvalitnění předkládaného učebního textu.

Obsah

Kapitola 1

Úvod

Chceme-li definovat, co znamená modelování a simulace, musíme před tím definovat význam některých výchozích termínů jako systém a model i termínů pomocných, které pomohou vyjasnit některé nepříliš přesné, avšak všeobecně rozšířené představy. Použité termíny jsou většinou cizí slova, takže je používají i jiné jazyky, zejména „mateřský jazyk počítačové profese“ – angličtina. Je záhodno být ve shodě s tímto jazykem, protože český počítačový odborník je a bude během své budoucí práce nucen číst řadu anglických knih a článků. Současně však je třeba vzít v úvahu, že termíny používané v simulaci a modelování mohou mít jiný význam v běžném jazyku, resp. v odborných terminologiích jiných profesí, a na nejdůležitější odlišnosti upozornit.

1.1 Systém

Slovo „systém“ je v dnešní době používáno v mnoha oborech a v mnoha významech. (Všimněme si např. významové vzdálenosti mezi výrazy operační systém a jazykový systém arabštiny.) Neodpovědným používáním v politice a masmédiích („systémové pojetí“, „systémové změny“, „systémový přístup“, ...) ztratil tento termín v běžném jazyku téměř všechen význam, stal se prázdnou frází. Jeden obor, který často používá simulaci, totiž teorie regulace a technického řízení, vymezuje termín systém dosti přesně (jako objekt se vstupními a výstupními signály svázanými přes své vnitřní stavy pomocí obyčejných diferenciálních nebo diferenčních rovnic), avšak tento fakt nás nesmí svést k tomu, že bychom v simulaci a v modelování chápali systém podobně. Tam jde o něco zcela jiného, jak dále vysvětlíme.

V simulaci a modelování se studuje nějaká věc, resp. možné varianty nějaké věci, při čemž slovo věc chápeme tak, jak jej chápou filozofové: je to nějaký objekt hmotného světa, a to buď objekt, který vskutku existuje (např. organismus konkrétní osoby, konkrétní továrna, krajina, škola atd.), nebo o kterém

uvažujeme, že by existovat mohl (např. stroj, budova či výrobní provoz, který by měl být realizován, nebo nemocný organismus dané osoby, o jehož terapii se uvažuje, ale definitivní rozhodnutí dosud nebylo formulováno). Věc chápou filosofové v její úplné složitosti (pokud existuje) nebo spolu se všemi nejasnostmi její existence (pokud se uvažuje o možnosti věc realizovat) a chápou i to, že není v lidských silách celou věc racionálně, tj. rozumovými prostředky, pochopit a zvládnout. Tak to chápou i různé obory vědy, techniky a řízení společnosti, a proto zavádějí na zkoumaných věcech abstrakce, které zanedbávají některé aspekty těchto věcí; zanedbané aspekty jsou vybrány tak, že aspekty, které zbývají, jsou daným vědeckým, technickým či společenským oborem zvládnutelné: mimo jiné, mohou o nich racionálně komunikovat pracovníci odpovídající vědecké, technické či společenské profese.

Takovou abstrakci budeme v modelování a simulaci nazývat systémem a podle charakteru profese, která systém na věci „vidí“, „zavádí“ či „definuje“, dostává systém i přívlastek: např. televizní přijímač je obvykle chápán jako elektronický systém, neboť i jeho běžný majitel ví, že si ho kupuje pro jeho elektronické vlastnosti (tedy pro vlastnosti, které patří do profese elektroniky), avšak bytový architekt ho tak chápat nemusí stejně jako např. notář sepisující dědictví po majiteli bytu; nebo železniční síť se běžně chápe jako dopravní systém, i když ekolog v ní může vidět systém jiný stejně jako — někdy v minulém století — stavbyvedoucí jejích složek. Na jedné věci lze tedy „vidět“ více systémů.

Abstrakce může nebo nemusí zanedbat význam času. Např. význam času v systémech železniční dopravy nelze běžně zanedbat, avšak konstruktér mapy železniční sítě České republiky k jízdnímu řádu roku 1997 zanedbává jak to, že se po jednotlivých tratích pohybují v čase vlaky, tak to, že železniční síť může měnit před rokem 1997 i po něm. Systém, v němž se od významu času abstrahuje, se nazývá **statickým systémem** (anglicky **static system**). Pokud se od významu času neabstrahuje, pak jen výjimečně se berou v úvahu i jeho vlastnosti, jak je poznává moderní fyzika. V drtivé většině oborů se čas chápe "newtonovsky", to jest jako v klasické fyzice, čili tak, že je smysluplné mluvit o tom, že dvě „události“ nastaly v systému současně nebo jedna z nich nastala dříve než druhá. Systém, jehož čas se nezanedbává a je přitom chápán takto „newtonovsky“, se v modelování a simulaci nazývá **dynamickým systémem** (angl. **dynamic system**). Simulace se jinými než dynamickými systémy nezabývá [22].

Množina okamžiků, v nichž dynamický systém existuje, se nazývá **časovou existencí** tohoto systému; protože v praxi nemá význam mluvit o jiných druzích existence a termín časová existence (dynamického) systému je dlouhý, mluví se krátce o **existenci** (**dynamického**) **systému**.

Existence dynamického systému je dána také abstrakcí: např. počítač při ně-

jaké akci (např. při řízení výrobního systému) chápeme jako systém existující jen během této akce, přestože jakožto věc existuje jistě před ní a pravděpodobně i po ní. Množina takových okamžiků nemusí být ani interval reálných čísel: např. pro odborníka zaměřeného na logické obvody jsou zajímavé jen časové okamžiky reprezentující hodinový puls daného počítače a přechodové fáze mezi nimi ho nezajímají, nebo pro makroekonoma mohou být důležitá jen data na koncích periodicky se opakujících účtovacích období a od toho, co se děje během těchto období, abstrahuje; pro oba odborníky systém existuje jen v konečné množině navzájem izolovaných časových okamžiků. Tak, jak se to dělá už dávno ve fyzice, lze časovým okamžikům jednoznačně přiřadit polohu na časové ose pomocí reálných čísel. Existence dynamického systému může být v principu jakákoliv neprázdná množina reálných čísel. V praxi jde vždy o množinu „dostatečně velikou“, což je ovšem mlhavý, ale srozumitelný pojem.

Dynamický systém je v každém okamžiku své existence v jistém **stavu** (angl. **state**). To, pro co jsme výše použili slova „událost“, je změna stavu dynamického systému. Poněkud nadneseně lze říci, že statický systém je stále v tomtéž stavu; nadnesené je to proto, že se z takového tvrzení nedá nic důležitého odvodit. Moderní fyzika nás učí, že nezanedbáváme-li v systému její poznatky o čase, nemá smysl mluvit ani o jeho stavu v daném čase ani o jeho existenci a událostí jsou neurčité. Jak už jsme však uvedli, právě poznatky moderní fyziky o čase se v modelování a simulaci neuplatňují.

V modelování a simulaci se chápe systém tak, že je složen z **prvků** (angl. **elements**). Mezním případem je, že systém má jediný prvek; tato praxe je však v simulaci poměrně vzácná (konkrétně vzato jen v některých případech, kdy se má simulovat systém definovaný odborníkem v regulaci). Běžně se systém rozkládá na více prvků: známe-li jejich chování, můžeme snadněji porozumět tomu, co se děje v celém systému. Prvky systému, tedy prvky abstrakce na nějaké věci, mohou odpovídat komponentám, které na věci nějak poznáváme fyzicky (např. její jisté prostorové složky — to je v praxi velmi častý případ), logicky (např. schopnosti dané věci či jejích složek), ale simulace neklade žádná omezení na způsob, jak rozklad provedeme; někdy se např. výhodně uplatní, když mezi prvky systému zahrneme i dvojice nebo seznamy jiných prvků téhož systému.

V dynamickém systému se může počet jeho prvků během jeho existence měnit: systém (např. biologický) může růst a smršťovat se, avšak v technických a ekonomických aplikacích jde nejčastěji o to, že prvky mohou do systému „vstupovat“ a systém „opouštět“. Takové prvky se nazývají **transakcemi** (angl. **transactions** nebo **temporary elements**). Ve skutečnosti takové prvky nevznikají, nýbrž přicházejí do systému z jeho „okolí“, a nezanikají, nýbrž systém opouštějí; avšak vzhledem k tomu, že systém je abstrakce, která našemu rozumu nahrazuje zkoumanou věc, abstrahujeme i od okolí, které sice pro danou

věc existuje, ale pro systém nikoliv. Jinými slovy, když je nějaká složka reality přítomna v prostředí, od kterého abstrahujeme, je to v naší abstrakci stejné, jako by neexistovala. Jako příklady transakcí lze uvést zákazníky vstupující do obchodního domu, pacienty přicházející do nemocnice, zakázky přicházející do výrobního podniku nebo vozidla vstupující do dopravního systému, který je — jakožto systém — abstrahován tak, že je vydělen ze svého okolí, z něhož do něj vozidla přijíždějí a kam jej vozidla opouštějí. Prvky, které jsou v dynamickém systému během celé jeho existence, se nazývají **permanentními prvky** nebo **aktivitami** (angl. **permanent elements** nebo **activities**).

Právě jsme zdůraznili, že když o systému uvažujeme, zanedbáváme vše, co do něho nezahrnujeme. Z toho plyne, že jestliže transakce dynamický systém jednou opustí, je „ztracena“ z našeho uvažování a nemá smysl mluvit o jejím návratu; jinými slovy, pokud bychom připustili, že transakce systém skutečně opustila a pak se vrátila, nemělo by být možno identifikovat, že jde o tutéž transakci. Pokud se identita transakce identifikovat má, pak tato transakce nemůže systém opustit, nýbrž v něm zůstává, snad nějak skryta a bez důležitosti na dění v systému, avšak nemůže opustit systém, musí být stále do naší abstrakce zahrnuta.

Prvky systému mají své vlastnosti, které se odborně nazývají **atributy**. Příkladem může být teplota ingotu v systému oceláren (jde o tzv. **aritmetický** nebo **reálný** atribut, protože nabývá aritmetických hodnot, reálných čísel), funkčnost stroje ve výrobním systému (jde o tzv. **booleovský** atribut, protože nabývá booleovských hodnot „ano“ a „ne“, konkrétněji řečeno „schopen pracovat“ a „v poruše“), nebo jméno zákazníka banky (jde o tzv. **textový** atribut, neboť nabývá textových hodnot). Atributy tedy přiřazují prvkům nějaké hodnoty a ty se u prvků dynamického sytému mohou v čase měnit.

Na první pohled je patrno, že stav dynamického systému v čase t by měl být dán prvky, které jsou v čase t v tomto systému přítomny, a hodnotami jejich atributů v tomto čase. Při bližším rozboru se však ukáže, že stav dynamického systému je ovlivněn i relacemi mezi jeho prvky: je-li na příklad daná zakázka ve výrobním systému zpracovávána na jeho jistém stroji, chápeme přirozeně takový stav výrobního systému za jiný než stav, v němž existuje třeba jen jediná odlišnost, a to ta, že je tatáž zakázka zpracovávána na jiném stroji; odlišná relace mezi zakázkou a strojem má na odlišnost stavů podstatný vliv.

V praxi simulace a dalších oblastí modelování se však relace v tom smyslu, jak se jim rozumí v matematice a v operačním výzkumu a jak jsme je právě na příkladě naznačili, nezavádějí. Nahrazují se **referenčními** atributy (ang. **pointers**), totiž atributy, které přiřazují prvkům systému jiné prvky. Např. vztah „zakázka M je právě zpracovávána na stroji P“ se reprezentuje jako „hodnota referenčního atributu *zpracovávaná zakázka* prvku P je rovna M“ nebo „hodnota referenčního atributu *zpracovávající stroj* prvku M je rovna

P“. Atributy, které nejsou referenční, se nazývají **standardní**, protože přiřazují prvkům „standardní“ hodnoty (reálná čísla, booleovské hodnoty, texty), přítomné v mnoha systémech, zatím co referenční atributy přiřazují prvkům jiné prvky téhož systému nebo výjimečně „nic“ (např. tehdy, když je zakázka ve výrobním systému, ale není právě zpracovávána na žádném stroji, nebo když daný stroj právě nezpracovává žádnou zakázku).

Změna hodnoty referenčního atributu znamená změnu konfigurace (struktury) dynamického systému. (V návaznosti na obecné chápání slov „struktura“, resp. „konfigurace“, „systému“ lze tato slova i v oboru simulace chápat jako souhrn všech referenčních atributů prvků systému.)

1.2 Model

Slovo „model“ se používalo v běžné řeči nejprve pro předlohu. V odborném jazyku doby před simulací a virtuální realitou zůstal z této praxe termín „funkční model“, a to pro první exemplář navrženého výrobku, který pracuje tak, jak by výrobek pracovat měl, přestože jiné vlastnosti výrobku (např. estetické) tento exemplář ještě nemá. Z této praxe vznikla i interpretace slova model pro něco zvláštního, nezvyklého či nákladného (např. hlavně před druhou světovou válkou používané termíny model klobouku, model automobilu apod.).

V modelování a simulaci je termín **model** použit pro analogii mezi dvěma systémy. Jednoduché příklady nabízí mapa (model části země na papíře), socha (model osoby, zvířete atd. v neživém materiálu) nebo dětský vláček (model skutečného vlaku ve zmenšeném měřítku). Vztah obou systémů – **modelovaného** a **modelujícího** je dán tím, že každému prvku P modelovaného systému je přiřazen prvek Q modelujícího systému, každému atributu g prvku P je přiřazen atribut h prvku Q a pro hodnoty atributů g a h je dána nějaká relace. Její charakter není nějak obecně omezen, ale v případě, že g i h jsou aritmetické atributy, bývá taková relace úměrnost, tolerance (mapa zobrazuje jen přibližně), kombinace úměrnosti a tolerance (např. rozměry složek a částí dětského vláčku jsou přibližně úměrné odpovídajícím rozměrům skutečného vlaku) apod.

Jsou-li modelovaný i modelující systém statické, říkáme, že daný model je **statický model**. V simulaci se však uplatní jen tzv. **simulační modely**, totiž modely, které splňují následující požadavky [22]:

1. Jejich modelující i modelované systémy jsou dynamické systémy.

2. Existuje zobrazení τ existence modelovaného systému do existence modelujícího systému; je-li tedy t_1 okamžik, v němž existuje modelovaný

systém M_1, je mu přiřazen okamžik $\tau(t_1) = t_2$, v němž existuje modelující systém M_2, a tak je zobrazením τ přiřazen i stavu $S_1(t_1) = \sigma_1$ systému M_1 stav $S_2(t_2) = \sigma_2$ systému M_2.

3. Mezi stavy σ_1 a σ_2 jsou splněny požadavky na vztahy mezi prvky a jejich atributy, jak jsme je výše popsali pro modely obecně; jako kdyby každému stavu σ_1 modelovaného systému odpovídal stav σ_2 modelujícího systému tak, že oba stavy jsou ve vztahu statického modelu.

4. Zobrazení τ je neklesající; pokud nastane stav s modelovaného systému před stavem s^* téhož systému, pak stav, který odpovídá v modelujícím systému stavu s, nastane před stavem, který odpovídá stavu s^*, nebo mohou oba stavy nastat v modelujícím systému současně (totiž v případě, že modelující systém není „tak kvalitní“, aby dokázal zobrazit všechny detaily v modelovaném systému), nikdy však nemůže být časové pořadí stavů v modelovaném systému a jim odpovídajících stavů v modelujícím systému přehozeno.

Požadavek 4 umožňuje tomu, kdo konstruuje modelující systém, aby se při tom nechal inspirovat vztahy kausality v modelovaném systému. Jestliže platí, že nějaké vlastnosti modelovaného systému implikují, že později nastane v tomto systému něco, co ovlivní jeho stav, lze tuto zákonitost napodobit i v modelujícím systému. Příkladem na takový kauzální vliv může být implikace, že když nějaký permanentní prvek je schopen obsloužit jen jednu transakci a žádají ho o obsluhu dvě transakce brzy po sobě, pak druhá z nich musí čekat ve frontě. Jiným příkladem je to, že když se hodnota nějakého aritmetického atributu mění v čase spojitě a je větší než jisté číslo, pak bude větší než toto číslo i v jistém následujícím časovém intervalu.

Model je tedy složitá struktura, která váže dva systémy, jejich prvky a jejich atributy, a v případě simulačních modelů i existence obou systémů. V běžné mluvě se však ustálila praxe, že pod slovem model se rozumí modelující systém. Tato praxe není úplně výstižná a přesná, protože nevystihuje, že model není jen systém, nýbrž že je je obrazem „něčeho“ a že to „něco“ zobrazuje „nějakým způsobem“. Místo termínu „modelovaný systém“ se používá slova **originál**.

V případě, že jde o simulační model, mluví se raději o systému **simulovaném** a **simulujícím** než o modelovaném a modelujícím. Analogicky k právě zmíněné nepřesnosti v praxi terminologie existuje praxe, že se na místě termínu simulující systém používá termín **simulační model** nebo také **simulátor**.

Termín simulátor nezavádí nepřesnost, a tak ho budeme také používat v těchto materiálech, přestože ho někteří američtí autoři používají v poněkud užším smyslu (např. jako trenažér).

1.3 Modelování

Angličtina je dosti odlišný jazyk od češtiny. Jedna její vlastnost, kterou těžko pochopí ti, jejichž mateřštinou je čeština, spočívá v tom, že druh slova není dán jeho tvarem, nýbrž jeho kontextem. Zatím co v češtině je slovo „model“ podstatné jméno a slovo „modelovat“ sloveso, může být v angličtině slovo model podstatným jménem, přídavným jménem i slovesem, a to podle toho, jaká slova jsou před ním a za ním. A stane-li se slovesem, může to být libovolné sloveso, odvozené z podstatného jména model: např. vytvářet model (přesněji: vytvářet modelující systém) či používat model, a obojí z různých důvodů; model (resp. modelující systém) může někdo vytvářet prostě jen proto, že má z takové práce potěšení, nebo proto, že se na jeho samotné konstrukci něčemu přiučí, nebo proto, aby ho později k něčemu použil; a pod slovem „použít“ se může skrývat např. použít k zjištění něčeho o originálu, použít k vlastnímu potěšení, použít k trénování práce s originálem, použít jako náhradu originálu v běžném životě atd. A to vše (a mnoho jiného) lze v angličtině shrnout pod sloveso **to model** (tedy **modelovat**).

V češtině dostalo slovo modelovat postupem doby několik významů. Jeden z nich je „dávat věci nový prostorový tvar“; např. geografové říkají, že řeka svou erosivní činností modeluje krajinu, nebo historik umění použil výrazu „malování bylo doplněno modelováním“, když chtěl říci, že v jisté oblasti se nejprve porcelánové nádoby zdobily malbou květinových motivů, ale později byly malované květiny naznačeny na povrchu i plasticky; i děti používají slova modelovat v podobném smyslu o své zábavě.

Slova model a modelovat byla v posledních desetiletích používána tak často a tak nezodpovědně, že ztratila téměř všechen význam. Tento proces proběhl i v řadě vědeckých oblastí, a tak dnes má metodologie vědy ve věci termínu modelování zcela nejasno. Neexistuje všeobecně přijatá definice a mezinárodní odborné akce spíše „mapují“, co vše se pod tímto termínem rozumí: ekologové, ekonomové, sociologové, chemici, astrofyzici, kosmologové, jaderní fyzikové, fyzikové pevné fáze, statistikové, logikové a další obory s námahou zjišťují, v čem se při porozumění termínům model a modelování shodují a v čem se různí.

V oblasti, které se dříve říkalo kybernetika a která se dnes zaplňuje systematickými aplikacemi výpočetní techniky, dominuje anglicky psaná literatura, které je nutno se přizpůsobit, a tedy si uvědomit, že modelovat (angl. **to model**) a modelování (angl. **modelling**, případně – od amerických autorů – **modeling**) může mít takřka neomezeně mnoho významů, jak jsme výše naznačili. Můžeme však ještě dodat, že tehdy, kdy jde o modelování ve smyslu **výzkumné techniky** (nebo — jak se často říká — **metody poznání**), je už obsah tohoto termínu vymezen jasněji, a to v následujícím smyslu [11]:

Podstatou modelování ve smyslu výzkumné techniky je ná-

hrada zkoumaného systému jeho modelem (přesněji: systémem, který jej modeluje), jejímž cílem je získat pomocí pokusů s modelem informaci o původním zkoumaném systému.

V tomto smyslu tedy platí, že vytvoříme model, v němž modelovaným systémem je zkoumaný systém, ale my budeme experimentovat s modelujícím systémem, při čemž cílem bude dozvědět se něco o modelovaném systému. Pokud by cílem bylo pouhé vytvoření modelu, resp. modelujícího systému, šlo by o modelování, ale jakožto zábavu a ne ve smyslu výzkumné techniky. Pokud by cílem bylo nahrazení modelovaného systému modelujícím systémem v reálném životě, šlo by o modelování ve smyslu vytváření protézy, a pokud by cílem experimentování bylo dozvědět se něco o modelujícím systému bez vztahu k sytému modelovanému, vypadl by model úplně „ze hry“ a šlo by vlastně jen o přímě experimentování s modelujícím systémem.

Modelování jakožto široký obor aplikací výpočetní techniky je takřka výhradně zaměřeno na to, co jsme výše akcentovali: na modelování ve smyslu výzkumné techniky. Je však vhodné uvědomit si bytostný fakt, že modelování v tomto smyslu není na aplikace výpočetní techniky omezeno. Modelující systém může být abstraktní matematická struktura (vzorec, ...) manipulovaná lidskou myslí a interpretovaná třeba na papíře, může to být fyzikální analogie (např. hydrodynamická analogie elektrického procesu nebo tzv. Bohrův model atomu) apod. Je ovšem pravda, že v dnešní době se z mnoha důvodů stále více uplatňuje ve funkci modelujícího systému výpočet na číslicovém počítači. Při tom je vhodné si uvědomit, že zde platí jistá analogie s automatizací opravdových pokusů (pokusů se zkoumaným objektem a ne se systémem, který ho modeluje): podobně jako moderní experimentátor nemusí zkoumaným objektem osobně manipulovat, ale může řadu pokusů automatizovat, jmenovitě pomocí řídícího počítače, který pokus či posloupnost pokusů řídí a vyhodnocuje, tak ani pokus s počítačově realizovaným modelujícím systémem nemusí být interaktivním dialogem mezi počítačem a operátorem, nýbrž může být naprogramován jako série automaticky řízených, za sebou následujících pokusů; na tomtéž počítači, kde je realizován modelující systém, může být realizována i automatická manipulace s tímto modelujícím systémem, obojí dokonce může být integrováno do jednoho výpočtu. Blíže se k celé věci (včetně příkladů) vrátíme, až budeme probírat, co je simulace.

1.4 Simulace

V obecné mluvě značí simulace předstírání nemoci, bezvědomí, duševní poruchy apod. Profesionálně vzato, tento význam slova simulace by měl patřit

někam do sociální psychologie. Pojem simulace, jak ho chápe aplikovaná informatika a kybernetika a jak ho chápou i ostatní obory, když aplikují výpočetní techniku, má však zcela jiný obsah. Stručně řečeno, v této oblasti je simulace chápána jako modelování ve smyslu výzkumné techniky, při němž použitý model je simulační. Zrekapitulujme (viz [22, 11]):

> **Simulace je výzkumná technika, jejíž podstatou je náhrada zkoumaného dyn. systému jeho simulátorem s tím, že se simulátorem se experimentuje s cílem získat informace o původním zkoumaném dynamickém systému.**

V tomto smyslu budeme termín simulace dále chápat. Všimněme si, že zde platí vše, co bylo řečeno výše o modelování jakožto výzkumné technice. Předně cílem simulace je získat informace o simulovaném systému, zatímco pouhá jeho náhrada simulátorem nestačí. Taková náhrada se někdy nazývá emulací – na příklad simulátor jednoho počítače P_1 realizovaný na počítači jiného typu je jakousi protézou, která nahrazuje P_1 – ten např. není pro nás dostupný, ale máme pro něj programy. Za druhé simulátor nemusí být realizován na číslicovém počítači, ale dnes je takto realizován stále častěji: číslicový počítač má výhody v tom, že je dálkově dostupný přes sítě, že se dá použít i k jiným účelům (čehož lze využít, když zrovna nemáme důvod použít ho k simulaci), že nekazí životní prostředí a nespotřebovává mnoho energie; nezanedbatelná výhoda číslicového počítače je i v tom, že výpočty na něm (a tedy i pokusy se simulátorem) lze reprodukovat.

Zdůrazněme, že aby šlo o simulaci, musí být cílem experimentů se simulátorem snaha dozvědět se něco o simulovaném systému. Když je simulátor realizován jako výpočet na číslicovém počítači, může se složkou simulace stát i experimentování se simulátorem, jehož cílem je získat informaci o něm samotném a ne o simulovaném systému: nastane to tehdy, když např. zjišťujeme, zda v příslušném programu není programátorská chyba nebo zda v něm není použita nevhodná numerická metoda; tento proces se nazývá **ověření správnosti modelu** nebo cizím slovem **verifikace modelu**.

Sloveso **simulovat** (angl. **to simulate**) budeme ve shodě s anglicky psanou odbornou literaturou chápat jako dělat simulaci (ve smyslu právě vymezeném). Odborník, který simuluje, je v angličtině nazýván **simulationist**, jednoduchý český termín neexistuje. Některá spojení jako např. statická simulace, která se dříve vyskytla tu a tam hlavně v německé literatuře, je nejlépe ignorovat, neboť dnes působí paradoxně a tedy neodborně (jako např. hranatá koule); stejně lze doporučit vyhýbat se opačným výrazům: např. dynamická simulace je stejný pleonasmus jako hranatá krychle.

1.5 Simulace na počítačích

Ve starších dobách byl simulátor realizován na speciálních zařízeních a podle nich dostávala příslušná simulace přívlastek: elektromechanická, hydrodynamická, mechanická, odporová, galvanická, analogová (pomocí analogových počítačů) a hybridní (pomocí hybridních analogo-číslicových počítačů). Dnes vytlačila všechny tyto druhy simulace, při níž je simulátor realizován na číslicovém počítači, tedy simulace **číslicová** (angl. **digital simulation**). V dalším výkladu půjde jen o ni, a tak budeme přívlastek číslicová vynechávat.

Nějaké přesnější vyjádření charakteru simulace podle výše zmíněných pravidel pro její přívlastek se dnes nepoužívá: nepíše se o simulaci osobněpočítačové, pentiové, síťové či multiprocesorové. Když je však jasno, že jde o simulaci číslicovou, spojuje se možnost vyjádřit něco přívlastkem s něčím zcela jiným, totiž s charakterem simulovaného systému. Jestliže ten je spojitý, tj. jestliže se hodnoty jeho atributu mění v čase jen spojitě, mluví se o **spojité simulaci** (angl. **continuous simulation** nebo **continuous system simulation**, tj. simulace spojitých systémů). Jestliže je simulovaný systém diskrétní, tj. nenastávají v něm spojité změny v čase, mluví se o **diskrétní simulaci** (v anglické literatuře se používá téměř výhradě termínu **discrete event simulation**, tedy simulace diskrétních událostí). Je-li simulovaný sytém tak říkajíc kombinovaný, to jest má-li jak vlastnosti typické pro spojité systémy tak vlastnosti typické pro diskrétní systémy, mluví se o **kombinované diskrétně-spojité simulaci** nebo častěji prostě o **kombinované simulaci** (angl. **combined simulation**).

Významy právě zavedených tří základních větví (číslicové) simulace jsou v různých situacích a různými autory více či méně modifikovány nebo i komoleny, proto potřebují blíže vysvětlit. Jelikož však jde o vskutku základní větve, bude každé z nich věnováno mnohem více než jen část této úvodní kapitoly, a tak bude bližší vysvětlení právě zavedených přívlastků formulováno v těch kapitolách, kde budou jimi označené základní větve rozvedeny.

Připomínáme, že systém je definován na věci. Na jedné a téže věci může být definován jak spojitý, tak diskrétní (případně i kombinovaný) systém. A tak se nesmíme divit, když je někdy jedna věc je zkoumána pomocí spojité, diskrétní a případně i kombinované simulace (např. odborník v oboru elektronických obvodů nebo polovodičové fyziky „vidí“ na počítači spojitý systém, a může tedy např. zkoumat procesor pomocí spojité simulace; avšak odborník v oboru hradlové logiky vidí na počítači diskrétní systém a může aplikovat na studium téhož procesoru diskrétní simulaci. Připomínáme dále, že i když simulujeme spojitý systém na číslicovém počítači, nesmí nás zmást fakt, že „uvnitř počítače“ existuje jakýsi diskrétní dynamický systém, vzniklý aplikací numerické metody a – jak se běžně říká – diskretizací modelovaného spojitého systému: i v takovém případě jde o spojitou simulaci, protože – jak už jsme uvedli –

přívlastek reflektuje to, jak my definujeme simulovaný systém, a ne to, co se děje v simulátoru.

1.6 Termíny používané při číslicové simulaci

Program, který řídí výpočet při (číslicové) simulaci, se nazývá **simulačním programem**. V jedné věci není ve světě vůbec jednoty, totiž zda se pod tímto termínem rozumí program ve strojovém kódu, který skutečně výpočet řídí, nebo program v programovacím jazyku, jak ho napíše jeho autor. Zdá se však, že důvod této nejednotnosti spočívá v tom, že v praxi kvůli ní nedochází k žádným fatálním nedorozuměním. A tak i v těchto materiálech budeme používat termínu simulační program jak pro text, který napíše autor simulačního modelu v programovacím jazyku, tak pro strojový kód, který z něho vznikne kompilací čili automatickým převodem do strojového kódu (interpretace zdrojového textu se dnes při simulaci pro svou zdlouhavost téměř nepoužívá).

Pokus se simulačním modelem se nazývá **simulační pokus** (angl. **simulation experiment**). V české literatuře se často vyskytuje termín simulační běh, ale ten nemá žádnou analogii v literatuře ve světových jazycích. Slovo **run** (tedy běh) se navíc hodí jako protiklad ke slovu kompilace, resp. překlad (např. „chyba zjištěná při překladu“ versus „chyba při běhu“), a – jak dále poznáme – při běhu neběží jen simulační pokusy.

Když na počítači běží simulační pokus, je záhodno evidovat při něm i čas, který by dané fázi výpočtu odpovídal v simulovaném systému, a to dejme tomu na adrese *time* (což je anglicky čas). Když je na této adrese hodnota T, pak výpočet jakoby sděloval „teď by měl být v simulovaném systému čas T“. Vzhledem k tomu, že v simulačním modelu nesmí být pořadí odpovídajících si stavů v simulovaném a simulujícím systému přehozeno, nesmí obsah adresy *time* během simulačního pokusu klesnout, nýbrž musí občas povyrůst. Mezinárodní autority v oboru simulace doporučily, aby se tyto hodnoty nazývaly **simulárním časem** (**simular time**), avšak odborná veřejnost používá ne zcela přesný, ale všeobecně rozšířený termín **simulovaný čas** (angl. **simulated time**).

Posloupnost simulačních pokusů majících stejný účel se nazývá **simulační studie** (angl. **simulation study**). V dnešní době je obvykle realizována jako jeden výpočet (task). Před každým pokusem se simulovaný čas vrátí zpět a rovněž se změní některé hodnoty způsobem, který nemá vzor v simulovaném systému (např. se vyprázdní fronty a vynulují některé součty). Navázání jednoho simulačního pokusu na druhý lze tedy nejlépe chápat jako analogii toho, že simulovaný systém zmizí a místo něho přijde v potaz systém zcela nový, který z historie původního systému „nedědí“ vůbec nic. Jeden „běh“ (ve smyslu použití zkompilovaného programu — viz výše) odpovídá tedy nejčastěji

jedné simulační studii, tedy několika simulačním pokusům (kdybychom použili výše zmíněného zlozvyku nazývat simulační pokus simulačním během, museli bychom přijmout bizarní tvrzení, že jeden běh se skládá z více simulačních běhů).

Ve světové literatuře se zavádí ještě termín **simulační krok** (angl. **simulation step**), a to pro časový úsek výpočtu, během něhož se nemění hodnota simulovaného času. Simulační studie se tedy skládá ze simulačních pokusů a ty se skládají ze simulačních kroků; na začátku každého pokusu se simulovaný čas vrátí na svou výchozí hodnotu (obvykle na hodnotu nula) a — s výjimkou prvního simulačního kroku každého simulačního pokusu — se zvětší hodnota simulovaného času o nějaký nezáporný přírůstek. Je-li tento přírůstek pro celý simulační pokus stejně velký, mluví se o **ekvidistantním** simulovaném čase, v ostatních případech se mluví o **neekvidistantním** simulovaném čase.

Kapitola 2

Mat. prostředky a metody pro modelování a simulaci

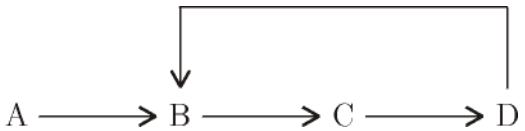
2.1 Matematické prostředky pro modelování a simulaci

Uvedeme jen některé významnější matematické prostředky používané při vytváření modelů.

Teorie množin a transformací se užívá především k popisu změn stavu systému (viz např. [3]). Předpokládejme, že jsme definovali množinu, jejíž prvky reprezentují všechny možné stavy systému, které se mohou realizovat v průběhu jeho vývoje. Tato množina nemusí být přirozeně konečmá, jejími prvky mohou být např. písmena nějaké abecedy. Dále zvolíme vhodný časový interval a ke každému ze stavů přiřadíme stav, do něhož by uvažovaný systém v průběhu zvoleného časového intervalu mohl přejít. Směr přechodu znázorníme šipkou. Takto vznikne model stavových změn (stavových přechodů), jenž kvalitativně popisuje vývoj sledovaného systému (viz schéma na 2.1).

Uvedené schéma můžeme též považovat za orientovaný graf, jehož uzly představují stavy systému a hrany možné stavové přechody.

Člověk se setkává se stále složitějšími objekty. Množství informace potřebné



Obrázek 2.1: Model stavových změn

k jejich popisu se přirozeně rovněž zvětšuje. Přesné modely složitých systémů se stávají nezpracovatelnými pomocí konvenčních matematických prostředků, proto je třeba hledat prostředky nové. Příčina tohoto stavu spočívá v tzv. principu inkompatibility (viz [53]). Roste-li složitost systému, klesá naše schopnost formulovat přesné a významné soudy o jeho chování, až je dosaženo hranice, za níž se přesnost a relevantnost prakticky vylučují.

Vhodným prostředkem pro popis nepřesných (vágních) pojmů je **teorie fuzzy množin** (např. [38]). Nejsme-li schopni přesně vymezit hranice nějaké třídy určené vágním pojmem, pak přiřadíme každému prvku míru jeho příslušnosti k dané třídě. Bude-li škála pro tuto míru uspořádaná, pak zřejmě platí následující tvrzení. Čím menší je míra příslušnosti daného prvku k uvažované třídě, tím blíže je tento prvek hranici třídy. Tato míra se nazývá stupeň příslušnosti prvku do uvažované třídy a třída, jejíž každý prvek je charakterizován stupněm příslušnosti do ní, se nazývá fuzzy množina. Teorie fuzzy množin lze využít i při zkoumání reálných systémů. Rozlišují se fuzzy systémy dvojího druhu:

1. skutečné fuzzy systémy, jejichž přesný teoretický popis neexistuje;

2. systémy, jež jsou natolik složité, že je nejsme schopni klasickými metodami přesně popsat.

Teorie fuzzy množin vytváří jakýsi most mezi verbálním a matematickým modelem reálného systému.

Významnou roli při vytváření modelů hraje **lineární algebra**, zejména **maticová algebra**. Matic se využívá především k matematickému popisu struktury systému a interakcí mezi jednotlivými prvky systému. Např. velikosti populací v n-složkovém systému můžeme popsat pomocí sloupcového vektoru $(x_1, x_2, \ldots, x_n)^T$ nebo řádkového vektoru $(x_1, x_2, \ldots, x_n)$, kde x_i značí velikost i-té populace. Interakce mezi jednotlivými populacemi se přehledně vyjadřují pomocí interakční matice (tabulky)

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \ldots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \ldots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \ldots & \alpha_{nn} \end{pmatrix}$$

v níž prvek $\alpha_{ij} \quad (i, j = 1, 2, \ldots, n)$ reprezentuje interakci mezi i-tou a j-tou populací. Maticová reprezentace je např. typická pro Leslieho diskrétní model rozvoje populace, jejíž natalita i mortalita jsou funkcí věku organismů.

Při konstrukci modelů se nejčastěji uplatňují **diferenční** nebo **diferenciální rovnice**. Pomocí těchto rovnic se simulují časové změny stavových pro-

měnných systému. **Diferenční rovnice** reprezentují změny, které se uskutečňují v průběhu diskrétních časových úseků. Je-li V_t hodnota určité stavové proměnné V v čase t, pak diferenční rovnice

$$V_{t+1} = f(V_t, t)$$

určuje hodnotu této proměnné (jako funkci původní hodnoty V_t a času t) po uplynutí časové jednotky. Pokud se dynamika systému systému popisuje soustavou diferenčních rovnic, V_t představuje vektor stavových proměnných v čase t a $f(V_t, t)$ onu soustavu rovnic. Diferenční rovnice jsou mimořádně vhodné pro vyjádření časových zpoždění, např.

$$V_{t+1} = f(V_t, V_{t-1}, \ldots, t).$$

Diferenciální rovnice popisují změny, které probíhají v čase spojitě, popř. kvazispojitě. Mají obvykle tvar

$$\frac{dV}{dt} = f(V, t),$$

přičemž V, $\frac{dV}{dt}$ i $f(V, t)$ mohou být chápány jako vektory. Diferenciální rovnice uvedeného typu vyjadřuje rychlost změny stavové proměnné V jako funkci okamžitých hodnot této stavové proměnné a času.

Vedle obyčejných diferenciálních rovnic a jejich soustav se při modelování systémů poměrně často setkáváme i s **parciálními diferenciálními rovnicemi** a jejich soustavami (např. při studiu dynamiky populací v nehomogenním prostředí).

V případě stochastických modelů se využívá **stochastických diferenciálních rovnic**, jež mají tvar

$$dV_t = f(V_t, t) + g(V_t, t)dW_t,$$

v němž $\{V_t, t \geq 0\}$ značí náhodný proces a dW_t diferenciál Wienerova procesu (zjednodušeně náhodné veličiny s normálním rozdělením, jejíž střední hodnota je nulová a rozptyl roven dt). Stochastické diferenciální rovnice se uplatňují např. v genetice a při modelování aktivity neuronů.

Kromě již zmíněných matematických prostředků se při modelování systémů často uplatňují:

- matematická logika (klasická logika, vícehodnotová logika, fuzzy logika, temporální logika);
- integrodiferenciální a integrální rovnice (např. při studiu dynamiky populaci se započtením zpoždění ve vzájemných interakcích);

- orientované grafy (při modelování transportních jevů);
- strukturní termodynamika (zobecněné termodynamické síly a toky při modelování transportu).

2.2 Matematické metody pro modelování a simulaci

Obsahem tohoto odstavce je stručná charakteristika vybraných matematických metod, resp. teorií, které se v současnosti významně uplatňují v oblasti modelování a simulace.

Pohybové rovnice systému mají často velmi jednoduchý tvar

$$\frac{d\mathbf{V}}{dt} = \mathbf{f}(\mathbf{V}), \tag{2.1}$$

kde $\mathbf{V}$ značí vektor stavových proměnných délky n a $\mathbf{f}$ zobrazení $R_n \longrightarrow R_n$. Řešením této rovnice na časovém intervalu T rozumíme zobrazení $\varphi : T \longrightarrow R_n$ takové, že pro všechna $t \in T$ platí

$$\frac{d\varphi(t)}{dt} = \mathbf{f}(\varphi(t)).$$

Dále se předpokládá, že každým bodem stavového prostoru prochází právě jedno řešení a že každé řešení je možno prodloužit na interval $(-\infty, \infty)$. Klasický přístup vyžaduje nalezení obecného řešení rovnice (2.1) ve tvaru $\varphi(t, C_1, C_2, \ldots, C_n)$, přičemž hodnoty konstant $C_1, C_2, \ldots, C_n$ se stanoví z počátečních podmínek. Získáme-li obecné řešení, máme úplný obraz o všech řešeních rovnice (2.1). Ze zkušenosti však víme, že (s výjimkou lineárních rovnic) existují jen velmi speciální třídy diferenciálních rovnic, jejichž obecné řešení lze rozumně vyjádřit.

Při studiu dynamiky systému popsaného rovnicí (2.1) nepotřebujeme zpravidla znát obecné řešení, úplně postačí informace kvalitativní povahy o chování systému v tzv. ustálených režimech. Informace tohoto typu poskytuje **kvalitativní teorie řešení diferenciálních rovnic** (viz např. [2, 46]).

Zavádí se pojem dynamického systému jako zobrazení

$$\phi : R \times R_n \longrightarrow R_n, \qquad \phi(t, V) = \phi_t(V),$$

které má následující vlastnosti:

- ϕ_0 je identita ($\phi_0(V) = V$);

- ϕ je pro každé $t \in R$ difeomorfismus (homeomorfismus, jemuž přísluší diferencovatelné inverzní zobrazení);

- pro všechna $t, s \in R$ platí

$$\phi_{t+s}(V) = \phi_s(\phi_t(V)).$$

Ze základních vět o spojitosti a diferencovatelnosti řešení diferenciální rovnice (2.1) vyplývá, že každá taková diferenciální rovnice generuje dynamický systém, v němž $\phi_t(V)$ představuje řešení této rovnice v čase t za předpokladu, že toto řešení vychází (v čase $t = 0$) ze stavu V. **Trajektorií (orbitou)** vycházející ze stavu V pak rozumíme množinu $\{\phi_t(V); t \in R\}$.

Vzhledem k tomu, že vektorová funkce **f** v rovnici (2.1) nezávisí explicitně na čase t, nezáleží vůbec na tom, ve kterém časovém okamžiku prochází řešení této rovnice stavem V. Zajímají nás jen takové případy, kdy každým bodem $V \in R_n$ prochází právě jediná orbita. Orbita může být:

1. bodem, jestliže platí $f(V) = 0$ (takový bod se nazývá **kritický bod**);

2. diferencovatelnou křivkou, která je uzavřená právě tehdy, když je řešení rovnice (2.1) procházející bodem V periodické.

Kritické body se interpretují jako rovnovážné stavy dynamického systému, kritické body spolu s **uzavřenými křivkami (periodickými orbitami)** jako **ustálené režimy** tohoto systému.

Kvalitativní teorie diferenciálních rovnic hledá především odpovědi na následující otázky (viz [5])).

1. Jaké jsou ustálené režimy sledovaného dynamického systému?

2. Je ustálený režim stabilní? Vrátí se systém po vychýlení z ustáleného režimu zpět nebo přejde do nějakého jiného ustáleného režimu nebo bude „bloudit“, aniž by se vůbec dostal do nějakého ustáleného režimu?

Všechny otázky kvalitativní povahy lze uspokojivě zodpovědět v případě, že známe rozložení orbit dynamického systému (tzv. **fázový portrét systému**). Představu o kvalitativní shodě fázových portrétů dvou dynamických systémů vyjadřuje nejlépe pojem topologické ekvivalence dynamických systémů. Dva dynamické systémy ϕ, ψ na množině R_n se nazývají **topologicky ekvivalentní**, existuje-li homeomorfismus $h : R_n \longrightarrow R_n$, jenž zobrazuje orbity systému ϕ na orbity systému ψ [5]. Kvalitativním řešením rovnice 2.1 rozumíme nalezení topologické struktury této rovnice, tj. zařazení rovnice do příslušné třídy topologické ekvivalence. Lokální topologická klasifikace diferenciálních

rovnic typu (2.1) v okolí kritických bodů i v okolí periodických orbit je uspokojivě vyřešena. Prakticky všechny možné diferenciální rovnice tohoto typu je možno zařadit do konečného počtu tříd, přičemž kritérium pro jejich zařazení je relativně jednoduché. Problém globální topologické klasifikace zůstal dosud nevyřešen. Doporučuje se provádět topologickou klasifikaci jen pro tzv. strukturně stabilní diferenciální rovnice. Rovnice (2.1) se nazývá strukturně stabilní, jestliže všechny rovnice z nějakého jejího okolí jsou s ní topologicky ekvivalentní. Nepřesnosti v zadání takové rovnice pak nemohou změnit kvalitativní chování příslušného dynamického systému.

V rámci kvalitativní teorie řešení diferenciálních rovnic se studují i rovnice závislé na parametru, tj. rovnice typu

$$\frac{d\mathbf{V}}{dt} = \mathbf{f}(\mathbf{u}, \mathbf{V}), \tag{2.2}$$

kde parametr $\mathbf{u}$ je obecně vektorem délky p a $\mathbf{f} : R_p \times R_n \longrightarrow R_n$. Je-li pro $\mathbf{u} = \mathbf{u}_0$ rovnice (2.2) strukturně stabilní, pak pro $\mathbf{u}$ z nějakého okolí $\mathbf{u}_0$ se topologická struktura této rovnice nezmění. Pokud však rovnice (2.2) není pro $\mathbf{u}_0$ strukturně stabilní, existují v libovolném okolí $\mathbf{u}_0$ rovnice typu (2.2) s různými topologickými strukturami. Taková změna topologické struktury rovnice (2.2) v důsledku změny parametru $\mathbf{u}$ se nazývá **bifurkace**. Problematice bifurkací je věnován přehledný článek [6].

Při modelování a simulaci některých jevů (např. šíření nervového vzruchu, psychické jevy) se s úspěchem používá Thomovy **teorie katastrof** [47]. K základním pojmům teorie katastrof lze dospět řešením soustav diferenciálních rovnic typu (2.2), kde vektorový parametr $\mathbf{u}$ popisuje vnější podmínky. Podobně jako v kvalitativní teorii řešení diferenciálních rovnic nás zajímají především ustálené režimy systému. Chování systému popsaného soustavou diferenciálních rovnic (2.2) je přirozeně závislé na hodnotě parametru $\mathbf{u}$. Při změně tohoto parametru dochází za jistých okolností v některých bodech, jímž se říká katastrofické body, k náhlé změně ustáleného režimu systému. Tyto náhlé změny (skoky) se nazývají katastrofami. Teorie katastrof se (řečeno zjednodušeně) zabývá řešením pohybových rovnic (2.2), ustálenými režimy vektoru $\mathbf{V}$ stavových proměnných, závislostí těchto ustálených režimů na hodnotách parametru $\mathbf{u}$ a zejména způsoby, jakými se může ustálený režim systému v katastrofických bodech měnit. Podstatnou kapitolu teorie katastrof představuje právě klasifikace těchto způsobů skokových změn (klasifikace katastrofických bodů). Z teorie katastrof je nejvýznamnější tzv. elementární teorie katastrof, jež zkoumá speciální (gradientový) případ, kdy

$$f_i(u, V_1, V_2, \ldots, V_n) = \frac{\delta P}{\delta V_i}$$

a P je nějaká reálná funkce definovaná na množině $R_p \times R_n$. Precizní výklad základních pojmů a tvrzení elementární teorie katastrof nalezne čtenář v článku [28], problémům aplikace teorie katastrof v přírodních a technických oborech je věnována monografie [48].

Zatímco teorie katastrof jako matematická disciplína je nepochybně významným oborem, její aplikace (metoda katastrof) je předmětem četných sporů. Používání pojmů a výsledků teorie katastrof má do značné míry heuristickou povahu. Všeobecně se soudí, že při seriózním modelování či simulaci je žádoucí kombinovat metodu katastrof s jinými matematickými prostředky.

K modelování a simulaci spojitých transportních jevů se často využívá **metody kompartmentových modelů (kompartmentových systémů)**. Klasické kompartmentové systémy je možno považovat za modely systémů hydrodynamických. Představují systém idealizovaných nádob (tzv. **kompartmentů**), jimiž proudí sledované látky (zpravidla směs nosiče a stopovací látky). Existují přirozeně i kanály, kterými látka proudí z okolí systému do některých kompartmentů (vstupy) nebo kterými opouští systém (výstupy). Objem kanálů se zanedbává, přitom se však předpokládá, že jimi může procházet nenulové množství látky v průběhu konečného (nenulového) časového intervalu. Každá nádoba je charakterizována vektorem atributů, jehož složky udávají množství (objemy) jednotlivých druhů látek a rychlosti jejich časových změn. O obsahu kompartmentů se předpokládá, že je homogenní (dokonale promíchaný). To znamená, že když do nějakého kompartmentu látka vstoupí, pak se okamžitě smísí s tím, co bylo v tomto kompartmentu dříve. Výhoda kompartmentových modelů spočívá v tom, že jsou to modely velmi názorné a jejich dynamiku lze poměrně jednoduše popsat analyticky pomocí nějaké soustavy obyčejných diferenciálních rovnic prvního řádu. Metody kompartmentových modelů se z počátku využívalo zejména v nukleární medicíně a radiobiologii. Později pronikly kompartmentové modely i do jiných oborů, např. do farmakologie a cytologie. Metoda kompartmentových modelů se přirozeně dále rozvíjí a v současné době se používá k modelování a simulaci i v epidemiologii a demografii. Obecná metoda multikompartmentových modelů je podrobně popsána v práci [30], matematizace jejích základních představ a principů je obsahem výzkumné zprávy [31]. Existují dokonce specializované simulační programovací jazyky pro analýzu kompartmentových systémů (např. jazyk COSMO [21]).

Pro zpracování velkých objemů dat (získaných např. zjišťováním anamnestických a diagnostických údajů na velkém počtu objektů) je určena **metoda automatizovaného generování hypotéz (metoda GUHA)** [16]. V tomto případě jde o aplikaci vyjadřovacích a deduktivních prostředků matematické logiky na analýzu empirických dat. Z formálního hlediska se analyzují data, která tvoří matematickou strukturu

$$\langle M, \varphi_1, \varphi_2, \ldots, \varphi_n \rangle,$$

v níž M je neprázdná konečná množina objektů a $\varphi_1, \varphi_2, \ldots, \varphi_n$ zobrazení typu $\varphi_i : M \longrightarrow V_i$, kde V_i jsou množiny hodnot sledovaných vlastností na jednotlivých objektech. Původně byla metoda GUHA navržena pro zpracování dvouhodnotových dat, tj. pro případ $V_i = 0, 1, i = 1, 2, \ldots, n$. V současné době jsou základní principy i matematická teorie metody GUHA formulovány natolik obecně, že jí lze užívat prakticky pro libovolný typ dat. Základním principem metody GUHA je syntaktické vymezení jisté třídy relevantních otázek, které mohou být na základě analýzy dat jednoznačně zodpovězeny. Tyto otázky jsou pak automaticky postupně generovány a zodpovídány na konkrétních datech. Metoda GUHA se neuplatňuje přímo v etapě vytváření modelu zkoumaného systému, ale především při testování hypotéz o zkoumaném systému. Je užitečná všude tam, kde jde o rozsáhlá experimentální data a jejich orientační analýzu (**exploratory analysis**). Pomocí metody GUHA se automaticky vytvářejí (vyhledávají) hypotézy o zkoumaném systému. Takové situace jsou běžné např. v klinických výzkumech a při komplexním studiu ekosystémů.

Významnou roli při studiu systémů hrají **metody matematické statistiky**. Těchto metod se využívá především v etapě ověřování hypotéz o studovaném systému na základě analýzy experimentálních údajů. Existuje přirozeně velmi bohatá a pestrá škála seriózních statistických metod, ne všechny se však uplatňují při verifikaci modelu ve stejné míře. V oblasti modelování a simulace se vedle testování statistických hypotéz nejčastěji uplatňují:

1. metody regresní a korelační analýzy,
2. analýza časových řad,
3. metody vícerozměrné statistické analýzy.

Úkolem **regresní analýzy** je nalezení vhodné teoretické regresní funkce k vystižení sledované závislosti, určení bodových, popř. intervalových, odhadů regresních koeficientů, určení odhadů hodnot regresní funkce pro účely prognostické a ověření souladu mezi navrženou regresní funkcí a experimentálními daty. Základním cílem **korelační analýzy** je měřit sílu (intenzitu, těsnost) korelační závislosti mezi sledovanými veličinami. V aplikacích jde většinou o vícenásobnou regresi a korelaci, protože se sleduje závislost vybrané náhodné veličiny na skupině dvou a více jiných veličin. Podrobné poučení o regresní a korelační analýze nalezne čtenář v monografii [54].

Problematika **analýzy časové řady** je detailně popsána v monografiích [1] a [8]. V praxi se nejčastěji vyžaduje vyrovnání dané časové řady a predikce hodnot sledované proměnné. Je zřejmé, že současné sledování několika časových řad přináší podstatně více informací než studium jediné řady. Vnitřní vazby mezi časovými řadami mohou totiž lépe osvětlit mechanismus vzájemné

interakce mezi sledovanými veličinami. Vícerozměrným časovým řadám je věnována např. monografie [17]. Při jejich analýze se řeší zejména dva okruhy problémů:

- posuzování charakteru a stupně závislosti mezi danými reálnými časovými řadami $\{X_t\}$ a $\{Y_t\}$,
- predikce ve vícerozměrných řadách (např. zlepšení předpovědi veličiny X na základě znalosti historie řady $\{Y_t\}$.

Z **metod vícerozměrné statistické analýzy** se při modelování a simulaci (v etapě ověřování hypotéz o zkoumaném systému) patrně nejvíce uplatňuje **shluková analýza** (viz např. [20]), která se zabývá klasifikací mnohorozměrných dat v situacích, kdy nemáme k dispozici žádný model, ale jen data samotná. Je dána nějaká množina $\{O_1, O_2, \ldots, O_N\}$ objektů, z nichž každý je reprezentován r-ticí hodnot atributů $\{a_{i1}, a_{i2}, \ldots, a_{ir}\}$, tj. bodem v r-rozměrném euklidovském prostoru. Úkolem shlukové analýzy je seskupit objekty do n shluků $S_1, S_2, \ldots, S_n$ tak, aby objekty patřící do téhož shluku si byly v určitém smyslu blízké, kdežto objekty, jež náležejí různým shlukům, co nejvíce odlišné. Důležitá je především vhodná volba míry podobnosti či nepodobnosti objektů, resp. shluků. Shlukovací procedury jsou v podstatě dvojího druhu: hierarchické a nehierarchické. V případě hierarchických postupů se shluky vytvářejí tak, že se vychází od jednotlivých objektů a ty se pak postupně shlukují na základě vhodné míry podobnosti (vzdálenosti), až se dospěje k požadovanému počtu shluků, resp. ke shluku jedinému. Nehierarchické postupy mají iterační charakter a jsou založeny na optimalizaci předem definovaného funkcionálu kvality rozkladu, který vyjadřuje matematické požadavky na stupeň podobnosti objektů uvnitř shluků, homogenitu rozložení objektů uvnitř shluků, rovnoměrnost rozložení objektů do různých shluků apod. Od hierarchických postupů se zásadně liší v tom, že připouštějí změnu rozmístění objektů do shluků v průběhu shlukování, a tím i opravu špatně zvoleného počátečního rozdělení. Oblast použití shlukové analýzy je mnohem širší, než se původně předpokládalo. Užívá se všude tam, kde jde o redukci dat s minimální ztrátou informace (např. při generování a ověřování hypotéz o zkoumaném systému, predikci založené na seskupování). Typické jsou např. aplikace v lékařské diagnostice (shlukování pacientů do homogenních tříd), ve farmaceutice (klasifikace pokusných zvířat) apod.

V sedmdesátých letech 20. století se začínají rozvíjet simulační metody založené na teoriích formálních jazyků a automatů. Na tomto místě uvádíme alespoň dva velmi známé přístupy:

- metoda založená na teorii **Lindenmayerových systémů** (**L – systémů**) [18],

- metoda založená na teorii **celulárních** (**buněčných**) **automatů** [19].

Obě uvedené metody původně vznikly s cílem modelovat (simulovat) evoluci mohobuněčných systémů. Vycházejí z následujících předpokladů:

- základní stavební jednotkou organismu je buňka,
- růst organismu probíhá v diskrétních časových okamžicích,
- změna stavu každé buňky v daném okamžiku je určena jejím vlastním aktuálním stavem a interakcí se sousedními buňkami,
- změna stavu celého organismu probíhá jako synchronizovaná změna stavu všech buněk.

Příklady na použití obou zmíněných metod jsou uvedeny v kapitole 6.

Uvedený přehled matematických metod a teorií užívaných v oblasti modelování a simulace nemůže být přirozeně úplný. Výběr těchto metod je do značné míry subjektivní, ovlivněný zaměřením autorů.

2.3 Základní fáze simulace

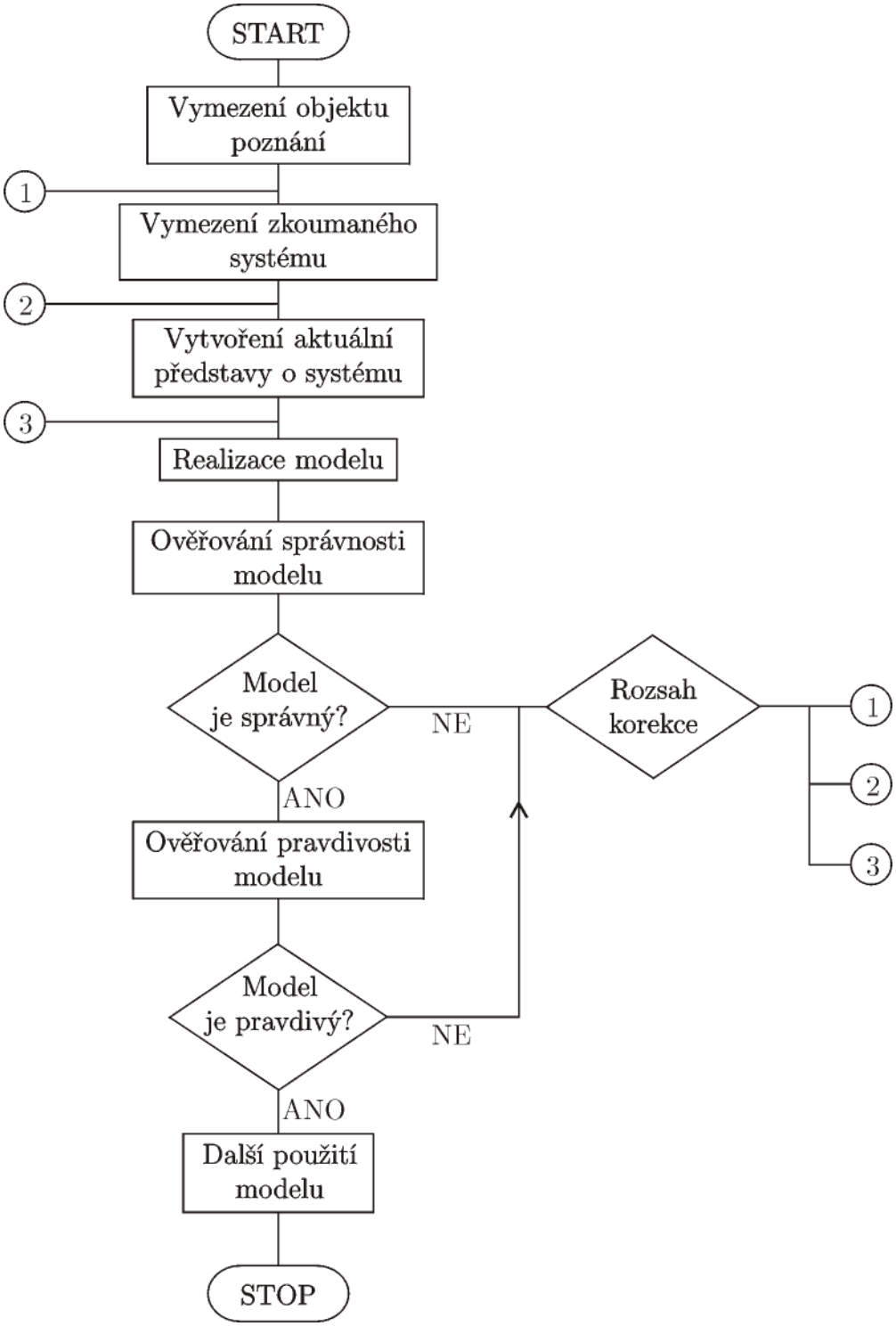
V tomto odstavci vycházíme z upřesněné Dohody o chápání pojmu simulace systémů [11], přijaté na půdě Komitétu aplikované kybernetiky ČSVTS.

Simulace systémů jako specifické formy procesu poznání se využívá při zkoumání i projektování objektů, dále při výuce, výcviku a v jiných případech sdělování poznatků a hypotéz. Předmětem simulace systémů jsou systémy vymezené na objektech poznání a jejich dynamika ve smyslu jakékoli změny v čase. Simulované systémy mohou být vymezeny jak na objektech již existujících, tak na objektech projektovaných. Připouští se i zkoumání systémů, které nemají bezprostřední vztah k objektivní realitě. Fundamentálním principem simulace systémů je vyvozování soudů o simulovaném systému na základě experimentů s jeho modelem (přesněji simulátorem).

Základní fáze simulace jsou zřejmé z vývojového diagramu na obr. 2.2.

Vymezením objektu poznání rozumíme vyčlenění zkoumaného objektu z okolního světa, resp. stanovení požadavků na projektovaný objekt a určení použitelných dílčích objektů ke konstrukci projektovaného objektu.

Máme-li definovat předmět poznání (simulovaný systém) jednoznačně, musíme určit především hledisko zkoumání daného objektu a zvolit odpovídající rozlišovací úroveň. Hledisko nazírání je dáno v první řadě účelem zkoumání daného objektu. V průběhu zkoumání objektu se ovšem rozlišovací úroveň (rozlišení) může s postupujícím poznáním měnit: zpravidla se zvyšuje, ale občas i snižuje, pokud poznáme, že je něco nadbytečné.



Obrázek 2.2: Vývojový diagram procesu simulace systému

Aktuální představa o simulovaném systému v sobě zahrnuje aktuální znalosti o zkoumaném systému, jeho struktuře a časových změnách, resp. zpracování projektu systému a identifikaci použitých subsystémů.

V etapě vytváření modelu jde o návrh simulujícího systému a jeho realizaci na vhodném simulátoru (nejčastěji na číslicovém počítači). Návrh modelu může, ale nemusí vycházet z matematického popisu aktuální představy o simulovaném systému. Za simulační se považuje jen takový model, jenž při napodobování dynamiky simulovaného systému zachovává uspořádání posloupnosti časových změn. V matematickém popisu modelu se rozlišují (viz např. [39]):

- stavové proměnné (veličiny reprezentující stav systému v kterémkoli okamžiku jeho existence);
- přenosové funkce (vztahy vyjadřující interakce mezi prvky systému, resp. mezi prvky systému a okolím);
- vynucující funkce (vstupní veličiny nebo faktory ovlivňující chování systému);
- parametry (konstantní veličiny charakterizující systém).

Model se považuje za správný, jestliže odpovídá aktuální představě o simulovaném systému. Ověřením pravdivosti modelu se rozumí verifikace hypotéz o zkoumaném systému, resp. ověření, zda vyprojektovaný systém splňuje stanovené požadavky a dá se realizovat. Z fází ověřování správnosti, resp. pravdivosti, modelu se v případě neúspěchu vracíme k fázím již absolvovaným. Způsob a rozsah korekce závisí přitom zejména na charakteru a závažnosti zjištěných nesrovnalostí.

Ověřeného modelu lze v procesu poznání využívat např. k identifikaci parametrů modelu, k prognózování, vědecké predikci, optimalizaci [50, 51], ale též ve výuce, výcviku apod.

Kapitola 3

Algoritmizace simulačního modelu

Je zřejmé, že pro formální matematický popis činnosti výchozího systému není prakticky možné navrhnout obecně platná pravidla. Určitě však má smysl pokusit se navrhnout jakousi univerzální metodiku pro algoritmizaci simulačního modelu (simulátoru) a jeho počítačové implementace. Taková univerzální metodika se přirozeně nemůže týkat těch částí algoritmizace, které jsou specifické pro konkrétní simulační úlohy, ale jen tzv. **simulačního jádra**, tj. té části, jež je prakticky společná všem simulačním programům. V této kapitole se budeme zabývat pouze takovými datovými strukturami simulačního jádra, které jsou společné pro simulační modely diskrétního i spojitého typu.

Hlavní úkoly při konstrukci simulačního jádra jsou (viz např. [33]):

1. navrhnout strukturu dat pro reprezentaci stavů simulovaného systému;
2. navrhnout operátory nad touto datovou strukturou, které realizují změny stavu systému;
3. zobrazit čas modelu a jeho průběh;
4. zajistit synchronizaci stavových změn systému tak, aby tyto změny probíhaly v určitém pořadí a při určitých hodnotách času nebo v okamžicích, kdy je splněna určitá podmínka týkající se stavu či konfigurace modelu.

3.1 Zobrazení stavů simulovaného systému

Pro reprezentaci stavů výchozího systému se používají nejrůznější datové struktury v závislosti na typu zvoleného programovacího jazyka. V nejjednodušším případě odpovídají skalárním stavovým veličinám jednoduché proměnné a vektorům datová pole.

Složitější datové struktury se využívají v případech, kdy se v průběhu času mění struktura systému nebo počet jeho prvků (např. při simulaci systémů hromadné obsluhy). V těchto případech je výhodné použít takových programovacích jazyků, jež umožňují přidělovat paměť exemplářům datových struktur dynamicky až v průběhu výpočtu

Objektově orientované programovací jazyky dovolují uživateli definovat (deklarovat) prakticky libovolně složité datové struktury a tyto dále obohacovat (koncepce tříd a podtříd). Např. při simulaci systémů hromadné obsluhy je velmi účelné použití spojových seznamů, které reprezentují fronty požadavků před obslužnými linkami.

3.2 Zobrazení stavových změn

Každá změna stavu je obecně realizována nějakým operátorem, který pracuje nad příslušnou datovou strukturou. Takový operátor sestává z posloupnosti příkazů, jejíž forma (makro, procedura, podprogram) je dána použitým programovacím jazykem. V moderních (objektově orientovaných) jazycích se tyto operátory logicky spojují přímo s odpovídajícími datovými strukturami (např. metody v deklaraci tříd jazyku Simula). Místo termínu „operátor“ používají někteří autoři slova „událost“, ale tuto praxi nedoporučujeme, protože termín „událost“ se používá v oblasti simulace v jiném významu.

3.3 Zobrazení času

Zobrazení času představuje specifický problém simulace. Čas výchozího (simulovaného) systému se zobrazuje v modelu pomocí aritmetické proměnné (buď typu **real** nebo typu **integer**). Pro tuto veličinu se zavádí speciální název - **simulární čas**. Nedoporučujeme používat v češtině dosti rozšířeného termínu „simulovaný čas“, ani termínů „modelový čas“, resp. „systémový čas“. Přitom musí být splněny určité podmínky:

- hodnota simulárního času nesmí v průběhu výpočtu klesat,
- děje v simulačním modelu závisejí na simularním čase stejným způsobem, jako jejich vzory ve výchozím systému na toku přirozeného času.

Hodnota simulárního času by neměla být uživateli přímo přístupná. V simulačních programovacích jazycích se zpravidla dodržuje následující zásada: uživatel může hodnotu simulárního času zjišťovat (např. prostřednictvím vhodné funkční procedury), nikoliv však měnit.

3.4 Synchronizace výpočtu

Stav dynamického systému v konkrétním časovém okamžiku je určen hodnotami jeho stavových proměnných. Má-li simulovaný systém n stupňů volnosti, bude jeho stav popisován hodnotami $s_1, s_2, \ldots, s_n$ stavových proměnných $S_1, S_2, \ldots, S_n$. Chceme-li v simulačním modelu sledovat vývoj takového systému, musíme v paměťovém prostoru vyhradit místo pro zobrazení jednotlivých stavových proměnných a pomocí vhodných programových prostředků zajistit aktualizaci hodnot těchto proměnných.

V případě diskrétní simulace předpokládáme, že se stav systému během konečného časového intervalu mění jen v konečně mnoha okamžicích. To znamená, že na spojité časové ose existuje pouze konečně mnoho časových okamžiků, v nichž se něco děje, v nichž nastávají tzv. **události** (**events**). Událostí přitom rozumíme změnu, jež je elementární a okamžitá (s nulovou dobou trvání). Podrobnosti nalezne čtenář v následující kapitole věnované algoritmizaci diskrétních simulačních modelů.

U spojitých dynamických systémů, které se obvykle popisují soustavou diferenciálních rovnic, se hodnoty stavových proměnných mění spojitě. Nicméně i v tomto případě se počítají hodnoty stavových proměnných v určitých diskrétních časových okamžicích daných velikostí kroku zvolené integrační metody. Tyto okamžiky pak mají pro spojitou simulaci stejný význam jako okamžiky, v nichž se realizují události v diskrétni simulaci.

Uvažujme simulační model, v němž nastávají události při hodnotách simulárního času $t_1, t_2, \ldots, t_m$. Chování modelu v časovém intervalu $\langle T_z, T_k \rangle$, kde T_z značí začátek a T_k konec simulačního experimentu, pak udává konečná posloupnost

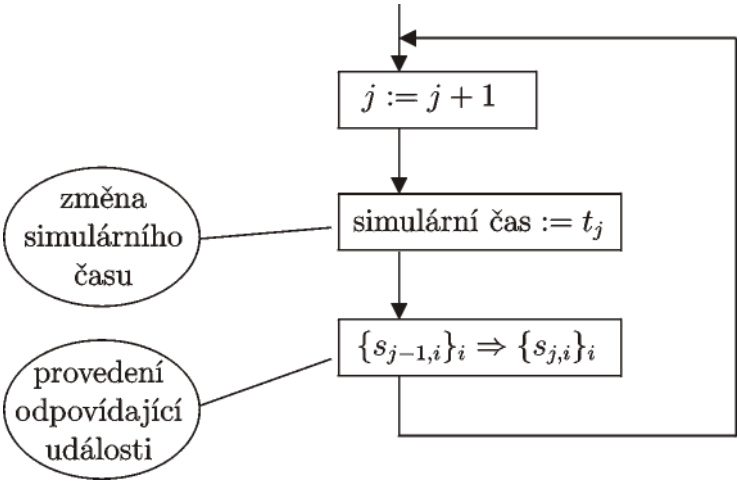
$$\{\{s_{j,i}\}_{i=1}^{n}, t_j\}_{j=1}^{m}$$

taková, že

1. $T_z \leq t_1, \quad t_m \leq T_k,$

2. $\forall j : 1 < j \leq m \Rightarrow t_{j-1} \leq t_j,$

3. posloupnost $\{s_{j,i}\}_{i=1}^{n}$ určuje stav modelu v simulárním čase t_j.



Obrázek 3.1: Schéma základního cyklu simulačního programu [33]

.

Změny stavu modelu se programově realizují pomocí posloupnosti příkazů, procedur nebo podprogramů. Požadované synchronizace mezi prováděním programové událostí a simulárním časem se dosáhne tak, že vždy po aktualizaci hodnoty simulárního času provedeme odpovídající programovou událost (viz schéma na obr. 3.1).

3.5 Vstupy a výstupy simulačních programů

Je-li simulačmí model výchozího systému zkonstruován a ověřeny jeho správnost a pravdivost, můžeme jej využít k vlastním simulačním experimentům. S organizací simulačních experimentů je spojena řada problémů. V tomto odstavci se budeme zabývat situací, kdy chování výchozího systému není deterministické. Konstruovaný model musí přirozeně tuto skutečnost respektovat (reflektovat), takže navrhovatel modelu musí specifikovat základní parametry stochastických procesů probíhajících ve výchozím systému.

Problémy související se stochastickým chováním modelovaných systému lze podle [33] rozdělit do tří skupin:

- specifikace parametrů rozdělení náhodných veličin,
- modelování procesů s náhodnými parametry,
- statistické hodnocení simulačních experimentů.

3.5.1 Specifikace parametrů rozdělení náhodných veličin

Pokud navrhujeme simulační model dosud neexistujícího reálného systému, nezbývá nic jiného než odhadnout tyto parametry na základě zkušeností s chováním obdobných reálně existujících systémů.

V případě, kdy reálný systém, který modelujeme, již existuje, můžeme jeho jeho chování sledovat po jistou dobu a potřebné parametry odhadnout na základě získaných experimentálních údajů. Experimentální data můžeme při specifikaci využít trojím způsobem:

- ke konstrukci teoretické distribuční funkce, tj. funkce dané exaktním vzorcem;
- ke konstrukci empirické distribuční funkce (zpravidla schodovité nebo po částech lineární), jejíž hodnoty se získají frekvenční analýzou experimentálních dat;
- přímo, pokud jsou získaná data dostatečně reprezentativní.

Pro experimentování se simulačním modelem se nejčastěji používá varianta první.

3.5.2 Modelování procesů s náhodnými parametry

Máme na mysli procesy, jejichž parametry mají charakter náhodných veličin. Získáme-li distribuční funkci takových parametrů jistého procesu, pak jej můžeme v modelu napodobit, tj. vytvořit proces, jehož parametry budou mít rozdělení dané získanou distribuční funkcí. Pro generování hodnot náhodných veličin v modelu se používají tzv. generátory pseudonáhodných čísel. Této problematice je ve skriptech věnován samostatný odstavec 4.4.

3.5.3 Statistické hodnocení simulačních experimentů

S vytvořenými simulačními modely provádíme experimenty, jejichž výsledky podléhají statistickému zpracování [27]. Přitom se požaduje, aby jednotlivé simulační experimenty byly statisticky nezávislé. Obecně platí, že výsledky simulačních experimentů s odlišnou (nezávislou) inicializací generátorů pseudonáhodných čísel uspokojivě splňují požadavky na nezávislost. Jak však zajistit nezávislost výsledků různých experimentů? V praxi se uplatňují čtyři odlišné přístupy (viz např. [33]):

1. Triviální je provedení série nezávislých simulačních experimentů, tj. série samostatných běhů simulačního programu s různou inicializací generátorů pseudonáhodných čísel.

2. V případě relativně stabilizovaného systému někdy postačí realizovat jediný simulační experiment a vybrat data, která jsou od sebe časově dostatečně vzdálena (tj. výsledky nezávislých úseků simulačního experimentu). Nezávislost vybraných úseků simulačního pokusu je nutno otestovat.

3. Při simulaci „malých“ systémů (malé nároky na paměť počítače) je možno postupovat tak, že se v simulačním programu modeluje hypotetický systém sestávající z více kopií výchozího systému, které pracují vzájemně nezávisle, samozřejmě podle stejných pravidel.

4. Speciální přístup je použitelný u tzv. **regenerativních systémů**, pro něž existuje taková rostoucí posloupnost časových okamžiků (tzv. **regenerativních okamžiků**) $\{b_i\}$, že chování systému mezi libovolnými dvěma po sobě následujícími okamžiky b_i a b_{i+1} je nezávislé na historii a všechny parametry ovlivňující chování systému v těchto intervalech mají identické rozdělení. V takovém případě lze chování systému v jednotlivých intervalech $\langle b_i, b_{i+1}\rangle$ považovat za nezávislé.

Kapitola 4

Algoritmizace diskrétních simulačních modelů

Diskrétní simulační modely jsou charakterizovány tím, že všechny stavové proměnné nabývají pouze diskrétních hodnot a v průběhu času se mění skokem. Nejčastějším případem diskrétních modelů jsou aplikace teorie hromadné obsluhy.

Pro diskrétní simulační modely jsou charakteristické následující rysy:

- proměnný počet prvků systému (požadavků),
- reprezentace front pomocí spojových seznamů,
- vysoký stupeň paralelnosti výpočtu,
- velké nároky na řízení programu (vyplývá z paralelnosti),
- vysoké nároky na paměť (velký počet prvků – požadavků).

V úvodní části této kapitoly jsou stručně vyloženy základy teorie diskrétních dynamických systémů. Zbývající část je věnována problematice výstavby diskrétního simulačního modelu s důrazem na konstrukci simulačního jádra programu (události, procesy, kalendář událostí, plánování procesů) a generování pseudonáhodných čísel.

4.1 Základy teorie diskrétních dyn. systémů

Teorie je propracována především pro případ jediné stavové proměnné, proto se i naše další úvahy omezí na tento případ.

Definice 1 Diskrétní dynamický systém *je matematická struktura určená třemi složkami:*

1. *intervalem I, v němž leží všechny možné hodnoty stavové proměnné x, např. $I = \langle a, b \rangle$, kde $a < b$;*

2. *funkcí f, definovanou na intervalu I;*

3. *diferenční rovnicí*

$$x_{n+1} = f(x_n),$$

jenž reprezentuje "pohybovou" rovnici uvažovaného systému.

Vyjdeme-li (v čase $t = 0$) z nějaké hodnoty x_0, potom posloupnost $x_0,\ x_1 = f(x_0),\ x_2 = f(x_1), \ldots$ nazýváme **trajektorie** bodu x_0, což není nic jiného než řešení dynamického systému s počátečním stavem x_0.

Pro libovolnou funkci f a libovolné celé kladné n můžeme zavést složené funkce f^n tímto předpisem

$$f^1(x) = x,\ f^2(x) = f(f(x)),\ \ldots,\ f^{n+1}(x) = f(f^n(x)).$$

Takto definovaná funkce $f^n(x)$ se nazývá **n-tá iterace funkce f**.

Mezi trajektoriemi daného dynamického systému zaujímají význačné postavení tzv. **stacionární trajektorie**, které odpovídají ustálenému pohybu systému. Jsou to:

- pevné body,
- periodické trajektorie (cykly).

Bod $\alpha \in I$ je **pevným bodem** funkce f, jestliže platí $f(\alpha) = \alpha$. Trajektorie pevného bodu α je zřejmě stacionární, všechny členy posloupnosti jsou totiž stejné (rovné α).

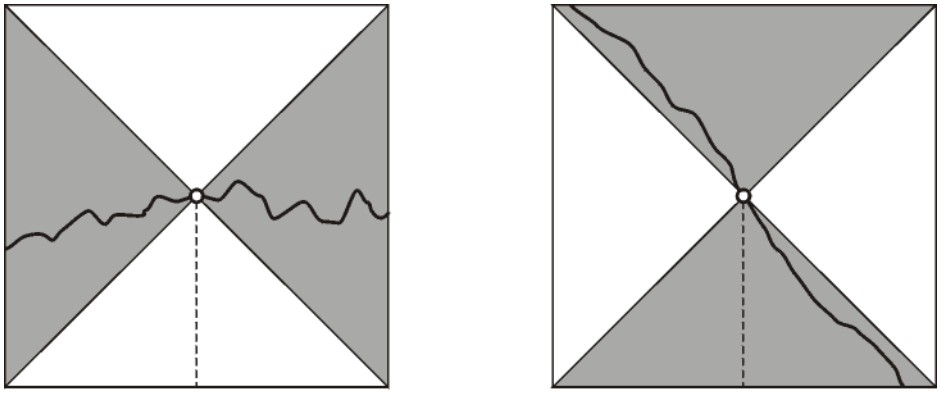
Pevné body funkce f se určí jednoduše: jsou to právě všechna řešení rovnice $f(x) = x$. Existují přirozeně různé typy pevných bodů.

Pevný bod α funkce f je:

- **atrahující (přitahující)**, jestliže existuje takový otevřený interval V obsahující α, že se trajektorie libovolného bodu $x_0 \in V$ (posloupnost $x_0, f(x_0), f^2(x_0), \ldots$) blíží postupně k α;
- **repulzivní (odpuzující)**, jestliže existuje takový otevřený interval V obsahující α, že trajektorie libovolného bodu $x_0 \in V$ ($x_0 \neq \alpha$) "vyběhne" po nějakém čase z intervalu V, tj. pro nějaké n platí $f^n(x_0) \notin V$.

Existují ovšem i takové pevné body, které nejsou ani atrahující, ani repulzivní.

Následující věta umožňuje rozhodnout, zda uvažovaný pevný bod α funkce f je atrahující či repulzivní.



a) b)

Obrázek 4.1: Ilustrace k určení typu pevného bodu

Věta 1 *Jestliže pro libovolné $x \in V$, $x \neq \alpha$ platí:*

1.

$$|\frac{f(x)-f(\alpha)}{x-\alpha}|<1, \tag{4.1}$$

pak α je atrahující pevný bod;

2.

$$|\frac{f(x)-f(\alpha)}{x-\alpha}|>1, \tag{4.2}$$

pak α je repulzivní pevný bod.

Podmínky (4.1) a (4.2) věty 1 je možno snadno interpretovat graficky (viz obr. 4.1). Uvažovaný pevný bod α je atrahující, jestliže graf funkce f (v jistém okolí bodu α) leží ve vyšrafované oblasti na obr. 4.1a, a repulzivní, pokud leží ve vyšrafované oblasti na obr. 4.1b.

Druhým případem stacionárního chování systému jsou periodické trajektorie, tvořené posloupnostmi, v nichž se jistý počet členů (tzv. cyklus) "do nekonečna" opakuje. Trajektorie tohoto typu sestávají z periodických bodů.

Bod $\alpha_0 \in I$ je **periodickým bodem řádu** n **funkce** f, jestliže platí $f^n(\alpha_0) = \alpha_0$, přičemž $f^j(\alpha_0) \neq \alpha_0$ pro všechna $j = 1, 2, \ldots, n-1$. Trajektorie takového bodu má tvar

$$\alpha_0,\ \alpha_1,\ \alpha_2,\ \ldots,\ \alpha_{n-1},\ \alpha_0,\ \alpha_1,\ \alpha_2,\ \ldots,\ \alpha_{n-1},\ \alpha_0, \ldots,$$

je to tedy periodická posloupnost s periodou n. Body $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}$ tvoří **cyklus řádu** n. (Pevný bod je podle této definice periodickým bodem řádu právě 1.)

Jak hledat periodické body řádu n pro danou funkci f? Také tato úloha je principiálně jednoduchá. V prvním kroku musíme nalézt všechna řešení rovnice

$$f^n(x) = x.$$

Z těchto řešení pak (ve druhém kroku) vyloučíme ta, jež představují pevné body funkce f a periodické body nižších řádů m (m je dělitelem n).

V praxi se často setkáváme s trajektoriemi, které sice nejsou přesně periodické (jako trajektorie periodických bodů), ale s postupem času se k nim přibližují. Takové trajektorie se nazývají **asymptoticky periodické**.

Z uvedeného je zřejmé, že má smysl mluvit o atrahujících a repulzivních cyklech.

Cyklus $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ řádu k funkce f je

- **atrahující**, když alespoň jeden bod tohoto cyklu je atrahujícím pevným bodem funkce f^k;
- **repulzivní**, když každý bod tohoto cyklu je repulzivním pevným bodem funkce f^k.

Právě uvedená definice zřejmě dovoluje rozhodnout o typu cyklu (jeho atrahovatelnosti či repulzivnosti) postupem, který jsme již vysvětlili v případě pevných bodů.

4.1.1 Ilustrativní příklad

Uvažujme dynamický systém určený funkcí

$$f(x) = Ax(1 - x), \tag{4.3}$$

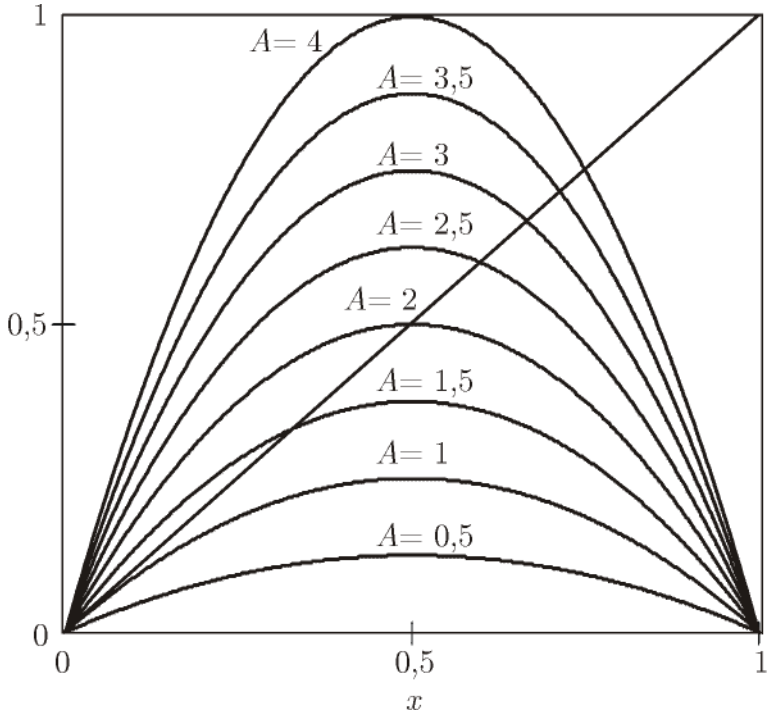
v níž stavová proměnná x reprezentuje relativní četnost nějaké populace v prostředí s omezenými zdroji ($x \in \langle 0, 1\rangle$) a A je reálný parametr charakterizující rychlost růstu uvažované populace ($A > 0$).

Pevné body funkce (4.3) získáme jednoduše řešením rovnice

$$Ax(1 - x) = x. \tag{4.4}$$

Pro přehlednost rozlišíme tři případy.

1. V případě $A \in \langle 0, 1\rangle$ má funkce (4.3) právě jediný pevný bod $\alpha = 0$ (viz obr. 4.2). Její graf leží celý pod grafem přímky $y = x$, takže uvedený pevný bod je atrahující. To ovšem znamená, že se trajektorie libovolného bodu $x_0 \in \langle 0, 1\rangle$ neomezeně blíží k nule. Biologická interpretace je jednoduchá: populace s tak malou rychlostí růstu zákonitě vymírá.



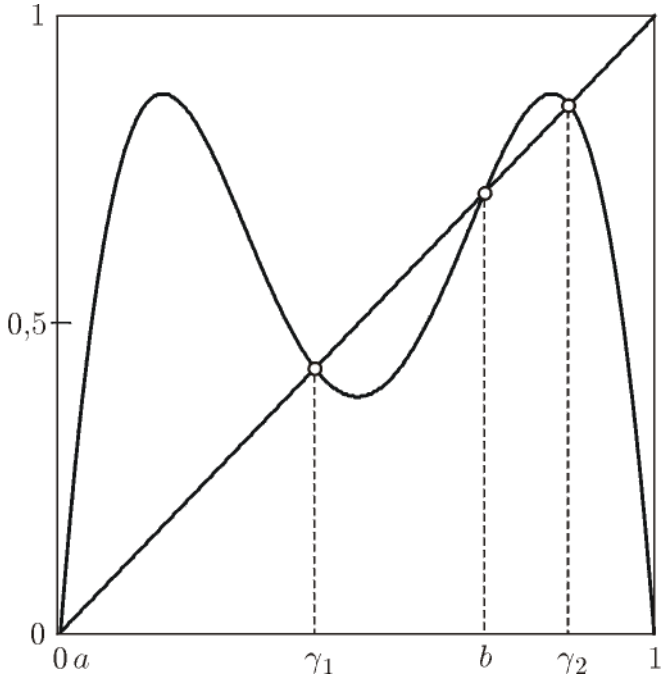
Obrázek 4.2: Ilustrace k pevným bodům funkce $Ax(1-x)$

2. Jak je patrno z obr. 4.2, pro $A \in (1,3\rangle$ existují právě dva pevné body funkce (4.3), a to $\alpha = 0$ a $\beta = 1 - 1/A$. Původně jediný pevný bod $\alpha = 0$ se "rozštěpí" na dva, přičemž nový pevný bod $\beta = 1-1/A$ se s rostoucí hodnotou parametru A vzdaluje od počátku. Můžeme snadno ukázat, že $\alpha = 0$ je nyní repulzivní pevný bod (graf funkce (4.3) v jeho okolí leží nad grafem $y = x$), kdežto $\beta = 1 - 1/A$ je atrahující pevný bod. Tuto skutečnost si můžeme ověřit numerickým výpočtem. Trajektorie libovolného bodu $x_0 \in (1,3\rangle$ se tedy přibližuje neomezeně k hodnotě $1 - 1/A$, což znamená, že relativní četnost populace postupně roste k uvedenému maximu (stavu nasycení).

3. V případě $A \in (3,4\rangle$ je situace velmi složitá. Při tak vysokých hodnotách růstového parametru A se mohou vyskytnout jak periodické trajektorie, tak i trajektorie asymptoticky periodické nebo dokonce zcela nepravidelné (chaotické).

Ukážeme, jak nalézt periodické body řádu 2 funkce (4.3), tj. trajektorii $\gamma_1, \gamma_2, \gamma_1, \gamma_2, \ldots$, kde $f(\gamma_1) = \gamma_2$ a $f(\gamma_2) = \gamma_1$. Zmíněné body najdeme řešením rovnice $f^2(x) = x$, tj.

$$A[Ax(1-x)][1 - Ax(1-x)] = x.$$



Obrázek 4.3: Ilustrace k určení periodických bodů řádu 2 funkce $Ax(1-x)$

Tuto rovnici lze přepsat ve tvaru

$$x[x-(1-1/A)][A^2x^2-(A^2+A)x+(A+1)]=0, \tag{4.5}$$

z něhož je zřejmé, že řešením (4.5) jsou oba již zmíněné pevné body $\alpha = 0$ a $\beta = 1 - 1/A$. Vyloučíme-li tato řešení, dostaneme obyčejnou kvadratickou rovnici

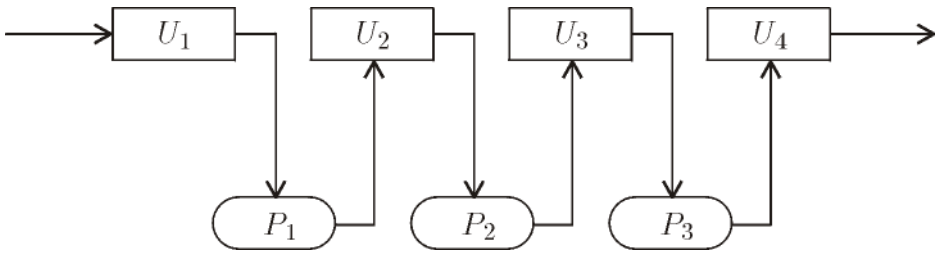
$$A^2x^2-(A^2+A)x+(A\ +1)=0. \tag{4.6}$$

Její diskriminant

$$D = A^4 - 2A^3 - 3A^2 = A^2(A+1)(A-3)$$

je pro $A > 3$ zřejmě kladný, takže rovnice (4.6) má právě dva kořeny γ_1 a γ_2 s požadovanou vlastností. Numerickým řešením např. pro $A = 3,5$ dostaneme $\gamma_1 = 0,42857$ a $\gamma_2 = 0,85714$.

Existenci těchto dvou periodických bodů γ_1 a γ_2 si můžeme názorně ověřit na obr. 4.3. Relativní četnost populace v takovém případě periodicky kolísá mezi hodnotami γ_1 a γ_2.



Obrázek 4.4: Obecné schéma procesu

4.2 Procesy

Na rozdíl od toho, co nabízí teorie, jsou diskrétní systémy, s nimiž se setkáváme v průmyslu, společnosti a komunikaci, obvykle vícerozměrné, dokonce s rozměrem, který se dynamicky mění v čase nebo je předem nepredikovatelný. V simulaci takových systému hraje klíčovou roli pojem procesu.

Při diskrétní simulaci existuje na spojité časové ose jen konečný počet okamžiků, v nichž nastávají změmy stavových veličin (události), které chápeme jako elementární a okamžité. Nositeli událostí jsou přirozeně objekty, přitom každá konkrétní posloupnost událostí je výsledkem složité interakce objektů. Většina událostí je takové povahy, že je lze zcela přirozeně vázat do složitějších celků, tzv. procesů (simulačních procesů). **Proces (process)** je tedy posloupnost logicky na sebe navazujících událostí. Strukturu procesu můžeme obecně popsat schématem uvedeným na obr. 4.4.

Proces se tedy neprovádí celý najednou (v jediném časovém okamžiku). V jednom určitém časovém okamžiku se tedy realizuje pouze část procesu odpovídající právě jediné události. Jednotlivé části procesu (události) U_i jsou od sebe odděleny tzv. plánovacími příkazy (příkazy potlačení procesu) P_i, jež způsobí, že se daný proces přestane provádět a začne se realizovat proces jiný. Při dalším vyvolání (aktivaci) se uvažovaný proces nezačíná provádět od začátku, ale od místa, kde byl naposledy přerušen.

Je-li tedy nějaký proces aktivován v jistém časovém okamžiku, provede se při této hodnotě simulárního času jen část uvažovaného procesu, a to právě událost, která se ve schématu na obr. 4.4 nachází bezprostředně za posledním realizovaným potlačením procesu. S potlačením procesu je přirozeně spojeno předání řízení výpočtu obecně jinému procesu. To umožňuje modelovat simultánně probíhající procesy pomocí sekvenčně prováděných instrukcí na jednoprocesorovém počítači.

4.2.1 Vnější stavy procesů

V souvislosti s plánováním procesů se rozlišují čtyři vnější stavy procesů:

1. **Stav aktivní (active)**. Proces je v aktivním stavu, je-li právě prováděn, tj. je-li právě realizován výpočet odpovídající některé jeho události. V aktivním stavu může být v daném okamžiku výpočtu nejvýše jeden proces.

2. **Stav ukončený (terminated)**. Proces se nachází ve stavu ukončeném, je-li je ukončena jeho operační část. Jestliže tato operační část obsahuje příkazy skoků, pak nemusí být zcela vyčerpána posloupnost událostí, z nichž se uvažovaný skládá. Takový proces již nemůže být ani aktivován, ani naplánován k provádění.

3. **Stav připravený neboli suspendovaný (suspended)**. Proces v suspendovaném stavu není sice aktivní, ale je naplánován k provádění v nějakém konkrétním okamžiku simulárního času. Pokud nebude jeho naplánování zrušeno jinými procesy nebo simulační pokus neskončí, dojde k jeho vyvolání.

4. **Stav pasivní (passive)**. V pasivním stavu je takový proces, který není ukončen, ale není také momentálně naplánován k provedení. K jeho vyvolání může dojít tehdy, bude-li naplánován prostřednictvím nějakého jiného procesu.

Jakýkoliv proces vytvořený v průběhu výpočtu simulačního programu se v každém okamžiku vyskytuje právě v jednom z uvedených stavů.

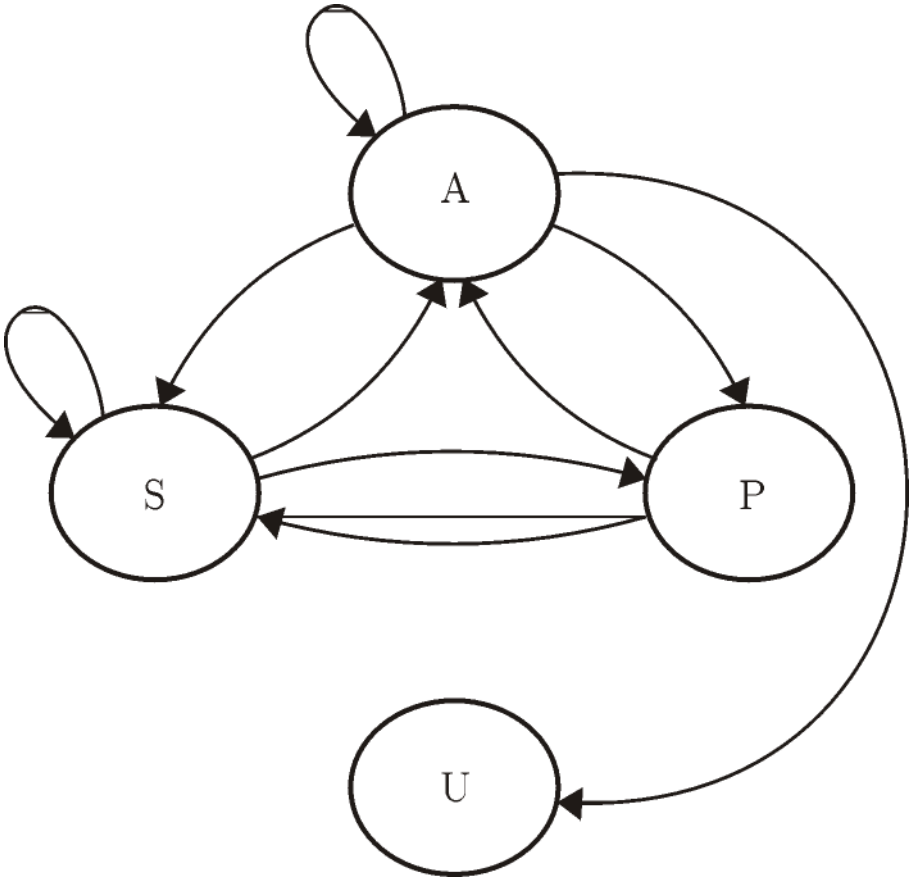
4.2.2 Změny vnějšího stavu procesu

Na obr. 4.5 jsou schematicky znázorněny všechny možné přechody mezi vnějšími stavy procesů [33].

Z toho, co bylo už dříve řečeno, je zřejmé, že přechody ze stavu ukončeného do jiných stavů jsou principiálně nemožné. Přechod procesu do ukončeného stavu se děje implicitně, jsou-li ukončeny výpočty v jeho operační části.

V této části se budeme podrobněji zabývat základními typy změn vnějšího stavu procesu.

Změna stavu aktivní $\longrightarrow$ suspendovaný. K této změně dochází tehdy, je-li potlačeno provádění právě aktivního procesu a pokračování tohoto procesu je naplánováno až po uplynutí jisté doby. V popsané situaci je možné, že nějaký jiný proces je naplánován k provedení v simulárním čase menším, než je čas pokračování dosud aktivního procesu, nebo ve stejném simulárním čase, ale s vyšší prioritou. V tomto případě se aktivním stane právě takový proces, který je naplánován k vyvolání nejdříve (proces s minimální hodnotou simulárního času, popř. proces, jenž je naplánován k vyvolání ve stejném čase jako pokračování původně aktivního procsu a má přitom nejvyšší prioritu).



Obrázek 4.5: Schéma možných přechodů mezi vnějšími stavy procesů

Ke změně stavu aktivní ⟶ suspendovaný může dojít i tehdy, když právě aktivní proces přímo vyvolá provádění procesu jiného.

Změna stavu aktivní ⟶ aktivní. V právě popsaném případě se ovšem může stát, že žádný jiný proces není naplánován s hodnotou simulárního času menší, než je hodnota, při níž má právě aktivní proces pokračovat, nebo se stejnou hodnotou simulárního času, ale vyšší priorotou než právě aktivní proces. Pak dojde pouze ke změně hodnoty simulárního času a dosud aktivní proces zůstává v aktivním stavu i nadále.

Změna stavu aktivní ⟶ pasivní. V podstatě jde o potlačení právě aktivního procesu, přičemž jeho pokračování není ještě explicitně naplánováno. Dosud aktivní proces setrvává v pasivním stavu, dokud jej nějaký jiný proces nepřevede do stavu suspendovaného nebo přímo aktivního. Podobně jako v předcházejících případech se po potlačení dosud aktivního procesu převede do aktivního stavu ten proces, jenž je naplánován k provedení nejdříve.

Změna stavu aktivní ⟶ ukončený. K této změně dojde při vyčer-

pání operační části uvažovaného procesu. Přitom se ukončí provádění tohoto procesu a řízení výpočtu se předá procesu, který je naplánován k provedení nejdříve. Při ukončení operační části nějakého význačného procesu může dojít k ukončení celého simulačního experimentu.

Změna stavu suspendovaný ⟶ aktivní. Taková změna může nastat dvojím způsobem:

- nepřímo při potlačení jiného procesu, který byl dosud aktivní;
- přímo tím, že zrušíme existující naplánování nějakého procesu a naplánujeme jej znovu s časem rovným momentální hodnotě simulárního času. Přitom nově aktivní proces se provádí při stejné hodnotě simulárního času jako dosud aktivní proces. Dosud aktivní proces je až druhý v pořadí, právě za nově aktivovaným procesem.

Změna stavu suspendovaný ⟶ pasivní. K této změně dojde jednoduše při zrušení již existujícího plánu uvažovaného procesu.

Změna stavu suspendovaný ⟶ suspendovaný. V tomto případě proces zůstává i nadále v suspendovaném stavu. Změní se pouze hodnota simulárního času, v němž je naplánována aktivace uvažovaného procesu.

Změna stavu pasivní ⟶ suspendovaný. Tato změna nastane, když naplánujeme budoucí aktivaci momentálně nenaplánovaného procesu.

4.2.3 Vnitřní stavy procesů

Poměrně složitý systém řízení simulačních výpočtů (potlačování právě aktivních procesů a aktivace jiných s minimálním časovým plánem aktivace) umožňuje na druhé straně zjednodušit jiné části simulačního programu. To se týká především zobrazování tzv. vnitřních stavů jednotlivých procesů. Významnou, nikoli však jedinou, charakteristikou vnitřního stavu procesu je jeho reaktivační bod. **Reaktivačním bodem** procesu přitom rozumíme právě to místo v operační části procesu, v němž je výpočet uvažovaného exempláře procesu přerušen a od něhož bude výpočet pokračovat při následující aktivaci tohoto exempláře.

4.3 Kalendáře událostí

Kalendář událostí nebo také **simulační kalendář** (např. v jazyku SIMULA **sequencing set**) je řídící struktura simulačního programu, která zahrnuje především programové prostředky pro plánování událostí. Posloupnost událostí v simulačním modelu není přirozeně dána předem. Proto je základním úkolem

simulačního programu tuto posloupnost vytvářet a průběžně aktualizovat. A je to právě kalendář událostí, jenž musí plnění tohoto úkolu zajišťovat.

Simulační programovací jazyky zpravidla již obsahují standardní prostředky pro plánování událostí. Pokud však implementujeme simulační model v nějakém jazyku, který nemá k dispozici standardní prostředky pro plánování událostí, musíme celé simulační jádro programu včetně plánovacích prostředků vytvořit sami.

4.3.1 Základní funkce kalendáře událostí

Vyžaduje se, aby kalendář událostí zabezpečoval čtyři základní funkce [33]:

1. zjistit, zda je daná událost (elementární část nějakého procesu) naplánována či nikoliv, a v případě, že naplánována skutečně je, zjistit hodnotu jejího aktivačního času;

2. vybrat proces s minimální hodnotou aktivačního času a pokud je takových procesů se stejnou hodnotou aktivačního času více, vybrat ten, který má nejvyšší prioritu;

3. naplánovat momentálně nenaplánovanou událost (proces);

4. zrušit plán momentálně naplánované události (procesu).

4.3.2 Návrh kalendáře událostí

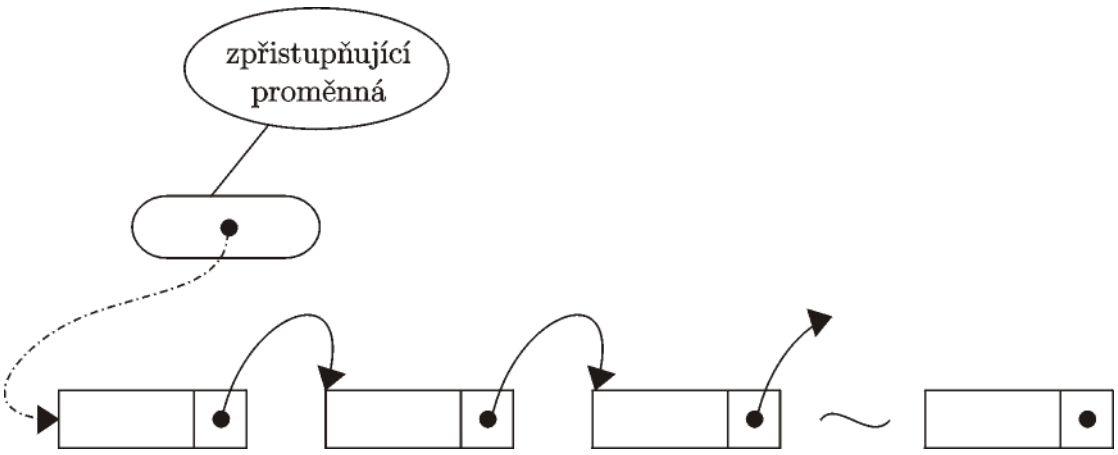
Datová struktura kalendáře událostí může mít samozřejmě celou řadu vzájemně odlišných implementací. Jednotlivé implementace se se liší především výpočetní složitostí a efektivitou provádění základních funkcí. Výběr implementace závisí přirozeně na charakteru řešené simulační úlohy.

Požadavkům na kalendář událostí vyhovuje libovolná struktura, která dovoluje:

- lineární uspořádání množiny právě všech událostí simulačního programu podle jejich aktivačních časů a stanovených priorit (existuje-li více událostí se stejným aktivačním časem),

- prohledávání této množiny podle číselného klíče (aktivačního času události) s přihlédnutím ke stanoveným prioritám.

Kalendář událostí je v podstatě uspořádaný seznam, jehož prvky obsahují informace o aktivačním čase dané události a její příslušnosti nějakému procesu.



Obrázek 4.6: Jednosměrný spojový seznam

Nejjednodušší implementací simulačního kalendáře jsou uspořádané lineární seznamy (lineární spojové seznamy). Právě takové implementace se využívá u většiny univerzálních, pro simulaci vhodných, jazyků.

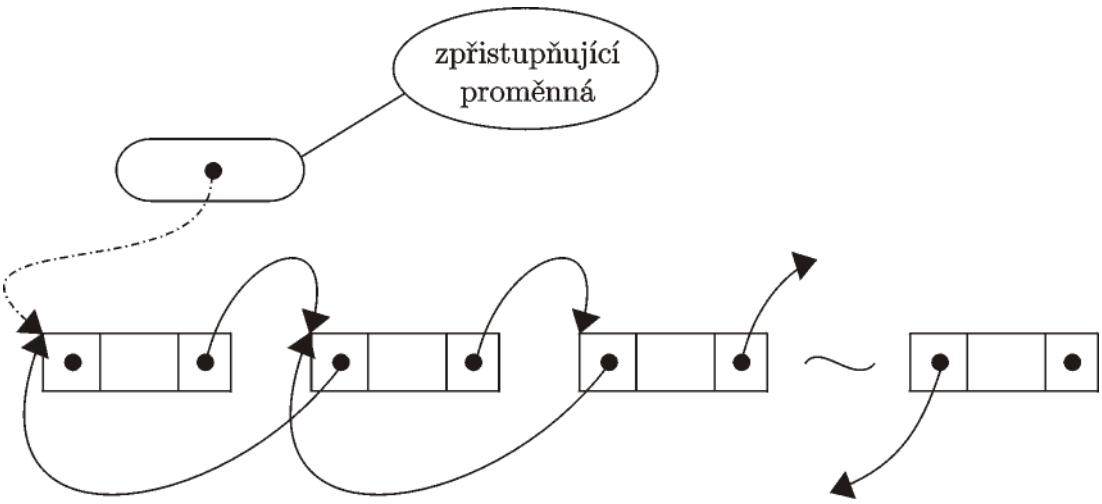
V případě lineárních spojových seznamů rozlišujeme:

- jednosměrné spojové seznamy,
- dvousměrné spojové seznamy.

Jednosměrné spojové seznamy jsou zpravidla zpřístupněny jedinou referenční proměnnou, která ukazuje na první prvek seznamu. Každý prvek pak odkazuje na svého následníka (viz obr. 4.6). Takové seznamy jsou vhodné ke konstrukci zásobníku pracujícího v režimu LIFO. Pro modelování fronty pracující v režimu FIFO lze také použít jednosměrného spojového seznamu, ovšem zpřístupněného dvěma referenčními proměnnými, z nichž jedna ukazuje na první a druhá na poslední prvek seznamu.

Jestliže však potřebujeme zařazovat prvky doprostřed seznamu (nejen na jeho začátek nebo konec) nebo vyřazovat prvky zprostředka seznamu, jsou jednosměrné spojové seznamy značně neefektivní, protože je třeba seznam pracně prohledávat. V takovém případě se doporučuje pracovat s **dvousměrným spojovým seznamem**. Takové seznamy (viz obr. 4.7) mohou být zpřístupněny jednou či dvěma referenčními proměnnými. Každý prvek seznamu (s výjimkou prvního a posledního) odkazuje nejen na svého následníka, ale také na svého bezprostředního předchůdce.

Podstatně efektivnější implementaci kalendáře událostí představuje **kruhový spojový seznam**. Jde v podstatě o speciální případ dvousměrného spojového seznamu, v němž první prvek je definitoricky následníkem posledního a poslední prvek předchůdcem prvního. Každý prvek seznamu bez výjimky má



Obrázek 4.7: Dvousměrný spojový seznam

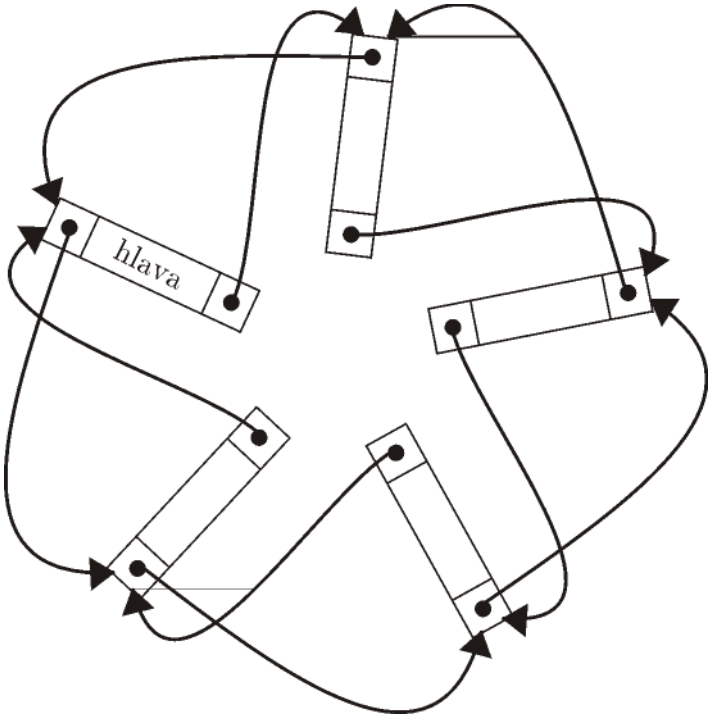
svého předchůdce i následníka a ke zpřístupnění seznamu stačí jediná referenční proměnná.

Problémy spojené se zařazováním prvního prvku, resp. s vypouštěním posledního prvku, se elegantně vyřeší tím, že se do seznamu vloží speciální prvek, který se mezi prvky seznamu nepočítá a který se nikdy nevyřazuje. Takový prvek se nazývá **hlava seznamu (head)**. Následníkem tohoto speciálního prvku je první prvek seznamu, jeho předchůdcem prvek poslední (viz obr. 4.8).

4.3.3 Hierarchické kalendáře událostí

Až dosud jsme předpokládali, že kalendář událostí obsahuje plány všech událostí uspořádané podle jejich aktivačních časů (s přihlédnutím k prioritám). Takové úplné uspořádání není ve skutečnosti nutné, protože vždy hledáme událost, jež má být aktivována jako první, a zpravidla nepotřebujeme znát, která událost je v pořadí druhá, třetí či poslední. Tato skutečnost vedla k myšlence konstruovat **hierarchický kalendář událostí**. Provede se rozklad množiny všech událostí simulačního programu (tzv. prostoru událostí) do několika, přirozeně neprázdných a disjunktních, podmnožin. Události je možno třídit podle prototypu události nebo podle struktury simulačního modelu. Např. pro různé části simulačního modelu vytvoříme samostatné kalendáře. Takové kalendáře mají samozřejmě menší rozsah a práce s nimi je podstatně efektivnější. V každém kroku výpočtu pak můžeme porovnávat jen hodnoty aktivačních časů prvních položek těchto samostatných (lokálních) kalendářů.

Pro hierachický kalendář můžeme také použít reprezentaci lineárního seznamu binárním stromem, což je běžná praxe v pokročilé programovací tech-



Obrázek 4.8: Kruhový spojový seznam (s hlavou seznamu)

nice [29].

4.3.4 Plánování událostí

Plánování jednotlivých událostí je obecně vázáno na splnění nějaké podmínky, nejčastěji dosažení určité hodnoty simulárního času, ale také dosažení určitého stavu modelovaného systému nebo určité konfigurace časových plánů jednotlivých událostí. V souvislosti s tím se rozlišují:

- **časové plánování** (také imperativní plánování nebo plánování vázaných událostí),
- **podmínkové plánování** (také interrogativní plánování nebo plánování podmíněných událostí).

V prvním případě se testuje pouze dosažení plánovaných hodnot simulárního času, ve druhém pak splnění podmínek týkajících se stavu modelovaného systému či konfigurace časových plánů událostí.

Kalendář událostí, který zajišťuje vedle časového i podmínkové plánování, musí obecně realizovat následující funkce:

1. určit, zda je daná událost naplánována či nikoliv a jakým způsobem je naplánována (časově nebo splněním nějaké jiné podmínky), a v případě, že je skutečně naplánována, specifikovat její aktivační čas, resp. tuto jinou podmínku;

2. vybrat takovou událost z podmínkově plánovaných událostí, jejíž podmínka je splněna (taková událost nemusí existovat);

3. vybrat takovou událost z časově plánovaných událostí, jejíž časový plán aktivace je minimální; existuje-li takových událostí více, vybrat tu s nejvyšší priorotou;

4. naplánovat momentálně nenaplánovanou událost a určit konkrétní čas její realizace, resp. podmínku její realizace;

5. zrušit plán momentálně naplánované události.

4.4 Generování pseudonáhodných čísel

Procesy v reálném světě mají často náhodný charakter. Zjistíme-li při analýze reálného systému náhodný charakter u některého z jeho procesů, musíme tuto skutečnost respektovat i při konstrukci simulačního modelu. Přitom zpravidla vycházíme z experimentálních dat. Na základě těchto dat můžeme

- buď vybrat vhodnou teoretickou distribuční funkci (exaktní vzorec popisující pravděpodobnostní rozdělení dat),

- nebo zkonstruovat empirickou distribuční funkci na základě rozdělení četností uvažovaných dat.

V praxi se hledají algoritmy vytvářející posloupnosti náhodných čísel, která splňují základní kritéria náhodnosti. Předpokládejme, že tato čísla jsou ve tvaru pravých desetinných zlomků s pevným počtem s číslic za desetinnou čárkou, tj. ve tvaru

$$\frac{a_1}{10} + \frac{a_2}{100} + \cdots + \frac{a_s}{10^s},$$

kde $a_i \in \{0, 1, \ldots, 9\}$, $i = 1, 2, \ldots, s$. Číslice a_i pak musí splňovat tyto podmínky:

1. každá číslice je vybrána náhodně z množiny $\{0, 1, \ldots, 9\}$, přičemž všechny prvky této množiny mají stejnou pravděpodobnost, že budou vybrány;

2. výběr číslice a_i nemá žádný vliv na výběr následující číslice a_{i+1}, $i = 1, 2, \ldots, s - 1$.

Protože s je konečné, pocházejí takto získaná čísla z rovnoměrného (diskrétního) rozdělení na intervalu $\langle 0, 1)$. Je-li však s dostatečně velké, můžeme toto rozdělení považovat za kvazispojité.

Posloupnosti náhodných čísel je možno generovat na základě mnohokrát opakovaných, skutečně náhodných experimentů (např. náhodný výběr prvků z množiny $\{0, 1, \ldots, 9\}$ s vracením vybraných prvků). V praxi se vytváření posloupnosti náhodných čísel řeší zpravidla softwarově pomocí speciálních algoritmů (viz dále), které v podstatě splňují základní kritéria náhodnosti. Taková čísla se označují jako **pseudonáhodná**, protože posloupnosti generované deterministicky (počítačem na základě daného algoritmu) nemohou být principiálně náhodné.

4.4.1 Algoritmy pro generování pseudonáhodných čísel

Historicky prvním algoritmem pro vytváření posloupnosti pseudonáhodných čísel je **algoritmus kvadratického středu**, který navrhl John von Neumann. Tento algoritmus je velmi jednoduchý.

0. vyber přirozené číslo o $2k$ číslicích,
1. umocni vybrané číslo na druhou,
2. vyber z mocniny prostředních $2k$ číslic,
3. přejdi do bodu 1.

Nevýhodou popsaného algoritmu je skutečnost, že po dostatečně dlouhé době dochází k vynulování generátoru.

V současnosti jsou zřejmě nejspolehlivější generátory založené na **multiplikativně kongruetní metodě**. Takové generátory vytvářejí z libovolného základního přirozeného čísla (násady generátoru) $x_0 < m$ posloupnost $\{x_i\}$ přirozených čísel z intervalu $\langle 0, m - 1\rangle$ podle rekurentního vztahu

$$x_{i+1} = (cx_i + h) \mod m,$$

kde c je násobitel, h přírůstek a m modul, přičemž $m > c$ a $m > h$. Generovaná posloupnost $\{x_i\}$ přirozených čísel je nutně periodická s periodou menší nebo rovnou modulu m.

Např. v programovacím jazyku SIMULA se používá rekurentního vztahu

$$U_{i+1} = U_i 5^{2p+1} \mod 2^n,$$

kde p a n jsou vhodně zvolené konstanty. Je-li výchozí U kladné liché číslo, pak jsou kladné a liché také všechny členy generované posloupnosti $\{U_i\}$. Posloupnost je periodická s periodou 2^{n-2} a při dostatečně velkém n splňuje velmi dobře základní požadavky kladené na pseudonáhodný výběr.

4.4.2 Generování pseudonáhodných čísel z daného rozdělení

Generátory popsaného typu vytvářejí pseudonáhodné posloupnosti $\{x_i\}$, jejichž členy pocházejí z rovnoměrného rozdělení na intervalu $\langle 0, 1)$. Pomocí jednoduché lineární transformace

$$y_i = A \ + (B - A)x_i \qquad (i = 1, 2, \ldots),$$

se pak generuje pseudonáhodná posloupnost z rovnoměrného rozdělení na intervalu $\langle A, B)$.

V praxi často potřebujeme generovat pseudonáhodná čísla z jiného než rovnoměrného rozdělení. Předpokládejme pro jednoduchost, že distribuční funkce požadovaného rozdělení $F(x)$ je spojitá a ryze monotónní, tj. existuje k ní funkce inverzní F^{-1}. V takovém případě platí:
Je-li $\{x_i\}$ posloupnost pseudonáhodných čísel z rovnoměrného rozdělení na intervalu $\langle 0, 1)$, pak posloupnost $\{y_i\}$, kde $y_i = F^{-1}(x_i)$, $(i = 1, 2, \ldots)$, je tvořena pseudonáhodnými čísly z rozdělení definovaného distribuční funkcí $F(x)$, protože

$$\mathbf{P}(y_i < z) = \mathbf{P}(F^{-1}(x_i)) < z) = \mathbf{P}(x_i < F(z)) = F(y).$$

Popsané metody (tzv. **inverzní metody**) nelze ovšem použít v každém případě. Je samozřejmě nepoužitelná tam, kde potřebujeme generovat pseudonáhodná čísla z nějakého rozdělení diskrétního typu (např. binomického nebo Poissonova). Také pro některé spojité distribuční funkce jsou vyvinuty speciální metody. Tak např. pro generování pseudonáhodných čísel z normálního rozdělení se využívá centrální limitní věty, podle níž má součet dostatečně velkého počtu náhodných veličin s rovnoměrným rozdělením na intervalu $\langle 0, 1\rangle$ přibližně normální rozdělení.

4.4.3 Testování generátoru

Před použitím generátoru v simulačních experimentech je nutno ověřit jeho vhodnost, což znamená, že skutečně generuje posloupnost čísel, která splňuje základní podmínky náhodnosti. Před vlastním testováním se generátor upraví tak, aby vytvářel posloupnost pseudonáhodných čísel z rovnoměrného rozdělení na intervalu $\langle 0, 1)$.

Testů navržených pro ověřování kvality generátorů je celá řada. Obecně platí, že je třeba k ověřování generátorů používat více testů, které hodnotí různé vlastnosti generované číselné posloupnosti. Nejčastěji se užívají:

- frekvenční test,

- testy autokorelace,
- sériový test.

Frekvenční test slouží k ověření rovnoměrnosti rozdělení generovaných pseudonáhodných čísel. Základní interval $\langle 0, 1)$ se rozloží na M stejně dlouhých subintervalů $\langle i/M, (i+1)/M)$, $i = 0, 1, \ldots, M-1$ a zjišťuje se počet generovaných čísel v jednotlivých subintervalech. Teorie říká, že při dokonale rovnoměrném rozdělení by měly být teoretické četnosti (počty generovaných čísel) ve všech subintervalech stejné. Frekvenčním testem se ověřuje shoda těchto teoretických četností s experimentálními četnostmi získanými při použití generátoru.

Testy autokorelace jsou určeny k hodnocení stupně vzájemné korelace (autokorelace) mezi členy posloupnosti generovaných pseudonáhodných čísel. Počítají se autokorelace řádů $1, 2, \ldots, m$, přitom se požaduje, aby tyto autokorelace nabývaly co nejmenších hodnot.

Sériový test patří k testům kombinovaným, které prověřují současně rovnoměrnost rozdělení i nezávislost. V principu jde o analogii frekvenčního testu. Provádí se frekvenční test ne jednotlivých pseudonáhodných čísel, ale m-tic po sobě jdoucích pseudonáhodných čísel.

4.4.4 Implementace generátoru pseudonáh. čísel

V simulačním programu jsou generátory pseudonáhodných čísel realizovány

- jednak procedurou, která provádí výpočet podle zvoleného algoritmu generátoru,
- jednak speciální proměnnou (tzv. **hnízdem generátoru** nebo také **násadou generátoru**), v níž se uchovává rekurentně měněná hodnota pro další volání procedury generátoru.

Jestliže potřebujeme simulovat více navzájem nezávislých náhodných procesů, je vhodné použít více různých hnízd generátoru. Každému procesu se pak přiděluje samostatné lokální hnízdo. Tato lokální hnízda musí být samozřejmě různě inicializována. Např. v jazyku SIMULA se k inicializaci těchto lokálních hnízd používají po sobě jdoucí lichá (kladná) čísla.

Kapitola 5

Algoritmizace spojitých simulačních modelů

Spojité simulační modely jsou charakterizovány tím, že všechny stavové proměnné nabývají hodnot z nějakého nedegenerovaného intervalu a v průběhu času se mění pouze spojitě. Chování (evoluce) spojitého modelu se popisuje pomocí soustavy diferenciálních rovnic, ať už obyčejných nebo parciálních. Simulární čas je diskretizován, a to tak hustě, aby numerické řešení diferenciálních rovnic splňovalo požadavky na přesnost výpočtů. Uchovává se jen okamžitá hodnota simulárního času, tato hodnota se postupně zvyšuje o délku integračního kroku.

Aplikace spojitých simulačních modelů přinášejí následující problémy:

- výběr vhodné metody numerické integrace,
- řízení délky integračního kroku s ohledem na požadovanou přesnost řešení úlohy,
- vysoké nároky na spotřebu strojového času.

Typické úlohy řešené pomocí spojitých simulačních modelů:

- kmitání mechanických systémů,
- řešení elektrických a elektronických obvodů,
- kompartmentové systémy,
- dynamika populací.

Úvodní část této kapitoly je věnována základům teorie spojitých dynamických systémů (viz např. [43]). Ve zbývající části jsou vyloženy principy nejčastěji používaných metod numerické integrace.

5.1 Základy teorie spojitých dynamických systémů

5.1.1 Definice spojitého dynamického systému a jeho řešení

Vyjdeme z rigorózní definice spojitého dynamického systému jakožto modelu nějakého reálného systému se spojitě se měnícími hodnotami atributů.

Definice 2 Spojitý dynamický systém *je matematická struktura tvořená:*

- *otevřenou množinou* $\Omega \subset \mathrm{R}^n$ *(tzv.* **stavovým prostorem***),*
- *množinou* $\{f_1, f_2, \ldots, f_n\}$ *reálných funkcí definovaných na* Ω*,*
- *soustavou obyčejných diferenciálních rovnic 1. řádu*

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \ldots, x_n), \tag{5.1}$$

kde $i = 1, 2, \ldots, n$.

Proměnné $x_1, x_2, \ldots, x_n$ se nazývají **stavové proměnné** dynamického systému a rovnice (5.1) **pohybové rovnice** tohoto systému. Přirozené číslo n nazveme **dimenzí** uvažovaného systému.

O stavovém prostoru se předpokládá, že je jistou otevřenou podmnožinou n-rozměrného euklidovského prostoru R^n . Okamžitý stav dynamického systému je pak reprezentován v tomto stavovém prostoru bodem, jehož kartézské souřadnice odpovídají hodnotám jednotlivých stavových proměnných.

Pro dynamický systém s pohybovými rovnicemi (5.1) platí, že okamžitá rychlost změny kterékoli stavové proměnné v daném okamžiku nezávisí explicitně na čase, ale pouze na okamžitých hodnotách stavových proměnných. Takovým dynamickým systémům říkáme **autonomní**.

Definice 3 Řešením dynamického systému *je množina reálných funkcí* $\{x_1(t), x_2(t), \ldots, x_n(t)\}$*, které mají následující vlastnosti:*

- *funkce* $x_i(t), i = 1, 2, \ldots, n$*, jsou definovány na nějakém nedegenerovaném otevřeném intervalu* $T \subseteq \mathrm{R}$*;*
- *derivace* $\frac{dx_i(t)}{dt}, i = 1, 2, \ldots, n$*, jsou také definovány na intervalu* T*;*
- *množina funkcí* $\{x_1(t), x_2(t), \ldots, x_n(t)\}$ *splňuje soustavu diferenciálních rovnic (5.1) v každém bodě* $t \in T$.

Křivky $C(t) = \{x_1(t), x_2(t), \ldots, x_n(t); t \in T\}$, které jsou určeny partikulárními řešeními daného dynamického systému, se nazývají **trajektorie** (**orbity**) tohoto systému. Množina všech trajektorií dynamického systému se nazývá jeho **fázovým portrétem**. Mezi trajektoriemi daného dynamického systému zaujímají zvláštní postavení:

1. **kritické body**, pro něž platí $x_i(t) = c_i$, přičemž konstanty c_i jsou řešeními soustavy rovnic

$$f_i(x_1, x_2, \ldots, x_n) = 0 \qquad (i = 1, 2, \ldots, n); \tag{5.2}$$

2. **periodické trajektorie** s **periodou** $\tau > 0$, které splňují pro všechna i, $i = 1, 2, \ldots, n$, podmínky

$$x_i(t) \neq x_i(0) \quad \text{pro} \quad 0 < t < \tau,$$

$$x_i(\tau) = x_i(0).$$

5.1.2 Existence a jednoznačnost řešení spojitého dynamického systému

V teorii spojitých dynamických systémů zaujímají význačné místo takové systémy, u nichž je zaručena existence právě jediného řešení. V následující větě jsou formulovány postačující podmínky pro existenci a jednoznačnost řešení dynamického systému.

Věta 2 *Nechť je dán spojitý dyn. systém s pohybovými rovnicemi (5.1). Nechť funkce $f_i(x_1, x_2, \ldots, x_n), i = 1, 2, \ldots, n$, splňují tyto podmínky:*

- *jsou spojité a omezené na otevřené množině $\Omega \subset \mathrm{R}^n$, přičemž $\mid f_i(x_1, x_2, \ldots, x_n) \mid \leq M$;*
- *vyhovují Lipschitzově podmínce na množině Ω, tj. existují taková reálná čísla $L_1^i, L_2^i, \ldots, L_n^i$, že platí*

$$\mid f_i(x_1, x_2, \ldots, x_n) - f_i(x_1', x_2', \ldots, x_n') \mid \leq \sum_{j=1}^{n} L_j^i \mid x_j - x_j' \mid .$$

.

Dále nechť je dán libovolný bod $[x_{10}, x_{20}, \ldots, x_{n0}] \in \Omega$ a číslo $t_0 \in \mathrm{R}$. Označme symbolem D vzdálenost tohoto bodu od hranice množiny Ω. Pak existuje právě jediné řešení $\{x_1(t), x_2(t), \ldots, x_n(t)\}$ uvažovaného dynamického systému, které je definované a spojité na otevřeném intervalu $(t_0 - D/(M\sqrt{(2)}), t_0 + D/(M\sqrt{(2)}))$, přičemž $x_i(t_0) = x_{i0} \quad (i = 1, 2, \ldots, n)$.

Důkaz této věty nalezne čtenář např. v učebnici [43].

Dynamické systémy, jež vyhovují podmínkám uvedené věty, jsou striktně kauzální. To znamená, že každý bod stavového prostoru Ω jednoznačně určuje celou trajektorii systému, právě tu, která jím prochází. Je-li dán počáteční stav takového systému, pak existuje právě jedna možná trajektorie, po níž se systém do tohoto stavu dostal, a také jediný možný způsob dalšího vývoje systému. Dále se budeme výhradně zabývat právě takovými systémy.

Veškeré dosavadní úvahy se týkaly autonomních dynamických systémů. Na první pohled se zdá, že předpoklad o časové nezávislosti chování dynamického systému je příliš restriktivní a nedovoluje nám řešit prakticky významné problémy. V modelech většiny reálných systémů vystupuje explicitně čas a navíc celá řada parametrů (např. rychlostní konstanty nebo vazebné koeficienty), které charakterizují daný systém a jeho dynamické vlastnosti. Zmíněné omezení je naštěstí jen zdánlivé a dynamické systémy s pohybovými rovnicemi (5.1) jsou natolik obecné, že zahrnují jak neautonomní systémy, tak i systémy s jedním nebo více parametry [43].

5.1.3 Stabilita řešení spojitého dynamického systému

Stabilita řešení dynamického systému souvisí s odezvou systému na poruchu (změnu počátečního stavu nebo hodnoty nějakého parametru), speciálně s limitním chováním trajektorií systému pro $t \to \infty$. Stabilita tedy charakterizuje globální vlastnosti trajektorie systému v přítomnosti poruchy ve vztahu ke globálním vlastnostem odpovídající trajektorie v nepřítomnosti poruchy.

Lokální informaci o odezvě dynamického systému na poruchu nám poskytuje tato věta [43].

Věta 3 *Nechť*

$$\frac{dx_1}{dt} = f_1(x_1, x_2), \frac{dx_2}{dt} = f_2(x_1, x_2)$$

jsou pohybové rovnice nějakého dynamického systému, který splňuje podmínky věty pro existenci a jednoznačnost řešení. Nechť $\{x_1(t), x_2(t)\}$ *je řešení systému takové, že* $x_1(t_0) = x_{10}$ *a* $x_2(t_0) = x_{20}$*. Pak existuje takové okolí* O *bodu* $[x_{10}, x_{20}]$*, že k danému* $\varepsilon > 0$ *můžeme nalézt* $\delta > 0$ *tak, že pro body* $[\bar{x}_{10}, \bar{x}_{20}]$ *splňující nerovnosti* $\mid \bar{x}_{10} - x_{10} \mid < \delta$, $\mid \bar{x}_{20} - x_{20} \mid < \delta$ *platí*

$$\mid \bar{x}_1 - x_1 \mid < \varepsilon \quad a \mid \bar{x}_2 - x_2 \mid < \varepsilon.$$

(Přitom $\{\bar{x}_1(t), \bar{x}_2(t)\}$ značí řešení daného dynamického systému, které prochází bodem $[\bar{x}_{10}, \bar{x}_{20}]$.)

Podle věty 3 je řešení dynamického systému, jež vyhovuje předpokladům věty o existenci a jednoznačnosti řešení, spojitou funkcí počátečních podmínek.

Analogicky lze dokázat, že taková řešení jsou spojitou funkcí jednoho nebo více parametrů. Tato tvrzení mají jen lokální platnost, tj. platí za předpokladu, že se omezíme na dostatečně krátký časový interval.

Analýza stability prakticky významných dynamických systémů ukázala, že existují tři základní typy chování systémů, pokud jde o vzájemný vztah mezi porušenými a neporušenými trajektoriemi. Uvažujme trajektorii $C(t) = \{x_1(t), x_2(t), \ldots, x_n(t)\}$ procházející bodem $[x_1(0), x_2(0), \ldots, x_n(0)]$. Předpokládejme dále, že přítomnost poruchy má za následek změnu počátečních podmínek, tj. přechod k porušené trajektorii $\bar{C}(t) = \{\bar{x}_1(t), \bar{x}_2(t), \ldots, \bar{x}_n(t)\}$, která prochází bodem $[\bar{x}_1(0), \bar{x}_2(0), \ldots, \bar{x}_n(0)]$.

Definice 4 Trajektorie $C(t)$ *se nazývá*

- **stabilní podle Ljapunova**, *jestliže ke každému danému* $\varepsilon > 0$ *existuje takové číslo* $\delta = \delta(\varepsilon) > 0$, *že pro* $t \geq 0$ *a všechna* $i \quad (i = 1, 2, \ldots, n)$ *platí*
$$| \bar{x}_i(0) - x_i(0) | < \delta \Rightarrow | \bar{x}_i(t) - x_i(t) | < \varepsilon;$$

- **asymptoticky stabilní podle Ljapunova**, *jestliže je stabilní podle Ljapunova a kromě toho pro všechna* $t > 0$ *a všechna* $i \quad (i = 1, 2, \ldots, n)$ *platí*
$$\lim_{t \to \infty} | \bar{x}_i(t) - x_i(t) | = 0;$$

- **nestabilní podle Ljapunova**, *jestliže existuje takové číslo* $\varepsilon > 0$, *že ke každému* $\delta > 0$ *můžeme nalézt aspoň jednu trajektorii* $\bar{C}(t)$ *a číslo* $t_1 = t_1(\delta) > 0$ *tak, aby platilo*
$$| \bar{x}_i(0) - x_i(0) | < \delta \quad \wedge \quad | \bar{x}_i(t) - x_i(t) | \geq \varepsilon$$
pro aspoň jedno $i \quad (i = 1, 2, \ldots, n)$.

Je zřejmé, že nějaká trajektorie $C(t)$ může být stabilní, resp. asymptoticky stabilní, jen když určitým způsobem omezíme třídu jí blízkých porušených trajektorií $\bar{C}(t)$. V takovém případě se hovoří o **podmíněné stabilitě**, resp. **podmíněné asymptotické stabilitě**.

Nechť $C(t)$ je trajektorie nějakého dynamického systému a S podmnožina jeho stavového prostoru Ω. Trajektorie $C(t)$ je **podmíněně stabilní**, resp. **podmíněně asymptoticky stabilní, vzhledem k množině** S, jestliže vztah porušených trajektorií $\bar{C}(t)$ procházejících body množiny S k uvažované trajektorii $C(t)$ odpovídá prvnímu, resp. druhému, případu v definici 4.

Poznámka. Zavedené pojmy stability se týkají odezvy dynamického systému na poruchu způsobenou změnou počátečních podmínek. V odborné literatuře

se setkáváme též s pojmem strukturní stability. Tento pojem souvisí s invariancí topologických vlastností trajektorií (topologií fázového portrétu) dynamického systému vůči malým změnám jeho pohybových rovnic.

Zvláštní postavení mezi trajektoriemi dynamického systému zaujímají **kritické (stacionární) body**. Jejich význam při studiu stability spočívá v tom, že obecný problém posouzení stability libovolné trajektorie systému může být převeden na úlohu posouzení stability kritického bodu nějakého jiného, zpravidla podstatně jednoduššího systému.

Uvažujme tedy dynamický systém s pohybovými rovnicemi (5.1) a předpokládejme, že existují jeho řešení typu $x_i(t) = \text{konst.}, i = 1, 2, \ldots, n$. Je-li počáteční stav systému určen podmínkami $x_i(0) = c_i, i = 1, 2, \ldots, n$, pak příslušná trajektorie představuje jediný bod ve stavovém prostoru systému, a to bod $[c_1, c_2, \ldots, c_n]$. Kritické body systému je možno považovat za degenerované trajektorie. Pro stabilitu kritických bodů platí v plném rozsahu definice 4, takže můžeme rozlišovat stabilní, asymptoticky stabilní a nestabilní kritické body (ve smyslu Ljapunova).

Následující věta (viz [43]) ukazuje, jak transformovat problém určení stability libovolné trajektorie daného dynamického systému na problém určení stability kritického bodu.

Věta 4 *Nechť dynamický systém s pohybovými rovnicemi (5.1) splňuje předpoklady věty pro existenci a jednoznačnost řešení. Nechť dále* $C(t) = \{x_1(t), x_2(t), \ldots, x_n(t)\}$ *je nějaká konkrétní trajektorie tohoto systému. Pak lze přejít k novým stavovým proměnným* $y_1, y_2, \ldots, y_n$ *(provést transformaci stavových proměnných) tak, aby platilo:*

- *pohybové rovnice uvažovaného systému v nových stavových proměnných mají tvar*
$$\frac{dy_i}{dt} = F_i(y_1, y_2, \ldots, y_n; t) \quad i = 1, 2, \ldots, n; \tag{5.3}$$
- *trajektorie* $C(t)$ *se transformuje do bodu* $[0, 0, ..., 0]$ *v soustavě nových stavových proměnných;*
- *počátek je kritickým bodem systému v nových stavových proměnných;*
- *stabilita počátku v nových stavových proměnných je táž jako stabilita trajektorie* $C(t)$.

Důkaz uvedené věty je triviální ([35]).

Výsledný dynamický systém s pohybovými rovnicemi (5.3) nemusí být autonomní, což znamená, že okamžitá rychlost jeho změny může záviset explicitně na čase. Tento dynamický systém se nazývá **systém poruch**. Jde v podstatě o původní dynamický systém reprezentovaný novými stavovými proměnnými y_i, $i == 1, 2, \ldots, n$. Právě uvedená věta nám umožňuje omezit se na analýzu

stability triviálních řešení $y_1(t) = y_2(t) = \ldots = y_n(t) = 0$ takových systémů poruch.

Jednoduchá a přitom účinná kritéria pro posouzení stability triviálního řešení (kritického bodu v počátku) poskytuje **Ljapunovova přímá (druhá) metoda**, která nevyžaduje integraci pohybových rovnic dynamického systému. Základní kritéria Ljapunovovy přímé metody budeme formulovat speciálně pro dvoudimenzionální dynamické systémy. Předpokládejme, že funkce f_1, f_2 jsou definovány na nějaké otevřené množině $\Omega \subset \mathrm{R}^n$ obsahující počátek, přičemž $f_1(0,0) = f_2(0,0) = 0$. Pak lze dokázat (viz [43, 36]) následující tvrzení.

Věta 5 *Jestliže existuje v Ω taková pozitivně definitní a spojitě diferencovatelná funkce $U(x_1, x_2)$, která má negativně definitní derivaci $\frac{dU(x_1,x_2)}{dt}$ podél trajektorií uvažovaného dynamického systému, pak je triviální řešení systému (kritický bod v počátku souřadnic) asymptoticky stabilní.*

Věta 6 *Jestliže existuje v Ω taková pozitivně definitní a spojitě diferencovatelná funkce $U(x_1, x_2)$, která má negativně semidefinitní derivaci $\frac{dU(x_1,x_2)}{dt}$ podél trajektorií uvažovaného dynamického systému, pak je triviální řešení systému stabilní.*

Věta 7 *Jestliže existuje v Ω taková pozitivně definitní a spojitě diferencovatelná funkce $U(x_1, x_2)$, pro níž platí $\frac{dU(0,0)}{dt} = 0$, a současně existuje alespoň jeden bod $[\varepsilon_1, \varepsilon_2] \in \Omega$ tak, že $\frac{dU(\varepsilon_1,\varepsilon_2)}{dt} > 0$, pak je triviální řešení uvažovaného dynamického systému nestabilní.*

Funkce $U(x_1, x_2)$, které vyhovují podmínkám předcházejících tří vět, se nazývají v uvedeném pořadí **Ljapunovovy funkce prvního, druhého**, resp. **třetího druhu**.

5.1.4 Stabilita řešení lineárních dynamických systémů

Relativně jednoduché je studium stability lineárních dynamických systémů, jejichž pohybové rovnice lze zapsat ve tvaru

$$\frac{dx_i}{dt} = \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, \ldots, n, \tag{5.4}$$

přičemž všechny koeficienty a_{ij} jsou reálná čísla. Počátek soustavy souřadnic (stavových proměnných) je evidentně kritickým bodem uvažovaného systému. Stabilita tohoto kritického bodu (triviálního řešení soustavy (5.4)) je určena, jak vyplývá z následující věty, vlastnostmi čtvercové matice $\mathbf{A} = (a_{ij})$, konkrétně jejími vlastními čísly λ_i, $i = 1, 2, \ldots, n$.

Věta 8 *Nechť je dán lineární dyn. systém s pohybovými rovnicemi (5.4). Nechť* λ_i, $i = 1, 2, \ldots, n$, *jsou vlastní čísla matice* $\mathbf{A}$. *Pak je kritický bod v počátku*

- *stabilní podle Ljapunova, když pro všechna* i, $i = 1, 2, \ldots, n$, *platí* $Re\lambda \leq 0$ *a zároveň všechna vlastní čísla s nulovou reálnou částí jsou jednoduchými kořeny charakteristické rovnice matice* $\mathbf{A}$;
- *asymptoticky stabilní podle Ljapunova, právě když platí* $Re\lambda < 0$ *pro všechna* i, $i = 1, 2, \ldots, n$;
- *nestabilní podle Ljapunova, když existuje aspoň jedno* i, $i = 1, 2, \ldots, n$, *takové, že platí* $Re\lambda > 0$.

Důkaz, jenž je poněkud zdlouhavý, nalezne čtenář v monografii [35].

Případ, kdy vlastní čísla s nulovou reálnou částí jsou vícenásobným kořenem charakteristické rovnice matice A, vyžaduje podrobnějšího rozboru (viz [35]).

Uvedená věta má základní význam pro posouzení stability uvažovaných lineárních systémů. Z této věty a z vlastností řešení soustav typu (5.4) vyplývá další významné tvrzení.

Věta 9 *Jakákoli trajektorie lineárního dynamického systému s pohybovými rovnicemi (5.4) je stabilní podle Ljapunova, resp. asymptoticky stabilní podle Ljapunova, právě když je jeho kritický bod v počátku ljapunovsky stabilní, resp. ljapunovsky asymptoticky stabilní.*

Pro posouzení asymptotické stability trajektorií dynamického systému s pohybovými rovnicemi (5.4) se často užívá **Hurwitzova kritéria**. Nechť polynom n-tého stupně

$$\lambda^n + h_1\lambda^{n-1} + \ldots + h_{n-1}\lambda + h_n$$

s reálnými koeficienty h_i, $i = 1, 2, \ldots, n$, je charakteristickým polynomem matice $\mathbf{A}$ koeficientů soustavy (5.4). Pak trajektorie odpovídajícího dynamického systému jsou asymptoticky stabilní tehdy a jen tehdy, když všechny hlavní diagonální minory tzv. Hurwitzovy matice

$$\mathbf{H} = \begin{pmatrix} h_1 & 1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ h_5 & h_4 & h_3 & h_2 & h_1 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & h_n & h_{n-1} & h_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & h_n \end{pmatrix}$$

příslušné danému polynomu jsou kladné, tj.

$$\Delta_1 = h_1 > 0, \Delta_2 = h_1h_2 - h_3 > 0, \ldots, \Delta_n = \det(\mathbf{H}) > 0.$$

Tabulka 5.1: Klasifikace kritických bodů 2-dimenzionálního lin. dyn. systému

Vlastní čísla matice **A**		Typ krit. bodu v počátku	Stabilita krit. bodu
Typ	Vztahy		
Reálné	$\lambda_1 = \lambda_2 < 0$	Uzel (propad)	Asympt. stabilní
	$\lambda_1 = \lambda_2 > 0$	Uzel (zdroj)	Nestabilní
	$\lambda_1, \lambda_2 < 0$ $\lambda_1 \neq \lambda_2$	Uzel (propad)	Asympt. stabilní
	$\lambda_1, \lambda_2 > 0$ $\lambda_1 \neq \lambda_2$	Uzel (zdroj)	Nestabilní
	$\lambda_1 < 0 < \lambda_2$	Sedlo	Nestabilní
Komplexně	$\alpha = 0$	Střed	Stabilní
sdružené	$\alpha < 0$	Ohnisko	Asympt. stabilní
$\lambda_1 = \alpha + \mathrm{i}\beta$	$\alpha > 0$	Ohnisko	Nestabilní
$\lambda_2 = \alpha - \mathrm{i}\beta$			

V praktických aplikacích se setkáváme nejčastěji s dvoudimenzionálními dynamickými systémy, proto považujeme za účelné uvést podrobněji klasifikaci kritických bodů pro takové systémy. Klasifikace kritických bodů těchto systémů je pro nenulová vlastní čísla λ_i, $i = 1, 2$, matice **A** uvedena v tabulce 5.1. Zvláštní zmínky zasluhuje případ, kdy jedna nebo obě vlastní čísla matice **A** jsou nulová. Platí následující tvrzení (viz [35]).

1. Jestliže $\lambda_1 = \lambda_2 = 0$, pak je každá trajektorie systému reprezentována jediným bodem ve stavovém prostoru. Trajektorie (body) takového systému jsou buď stabilní (ovšem neasymptoticky), jsou-li všechny prvky matice **A** nulové, nebo nestabilní, je-li aspoň jeden z prvků matice **A** různý od nuly.

2. Jestliže je pouze jedno z vlastních čísel λ_1, λ_2 matice **A** nulové, pak se dvoudimenzionální dynamický systém redukuje na systém jednodimenzionální. Ve stavovém prostoru takového systému existuje nikoli jeden, ale nekonečně mnoho kritických bodů, které určují přímku v tomto prostoru. Tyto kritické body jsou asymptoticky stabilní, resp. nestabilní, podle toho, zda to vlastní číslo matice **A**, jež je nenulové, má hodnotu zápornou, resp. kladnou.

5.1.5 Stabilita řešení nelineárních dyn. systémů

Uvažujme dynamický systém s pohybovými rovnicemi (5.1), který je definován na nějaké otevřené množině $\Omega \subset \mathrm{R}^n$ obsahující počátek a který splňuje předpoklady věty pro existenci a jednoznačnost řešení. Jestliže funkce $f_i, i = 1, 2, \ldots, n$, mají derivace všech řádů v každém bodě množiny Ω, můžeme pravé strany pohybových rovnic rozvinout v Taylorovu řadu se středem v počátku

$$f_i(x_1, x_2, \ldots, x_n) = f_i(0, 0, ..., 0) + \sum_{j=1}^{n} \frac{\partial f_i(0, 0, ..., 0)}{\partial x_j} x_j + \text{členy vyšších řádů.}$$

V dostatečně malém okolí počátku můžeme zanedbat členy druhého a vyšších řádů, takže pohybové rovnice uvažovaného dynamického systému dostanou tvar

$$\frac{dx_i}{dt} = b_i + \sum_{j=1}^{n} a_{ij} x_j; \quad i = 1, 2, \ldots, n,$$

kde $b_i = f_i(0, 0, \ldots, 0)$ a $a_{ij} = \frac{\partial f_i(0,0,...,0)}{\partial x_j}$. Dynamický systém s těmito pohybovými rovnicemi je lineární, ale jeho pohybové rovnice jsou nehomogenní. Nicméně vhodnou transformací stavových proměnných můžeme tyto pohybové rovnice převést na tvar (5.4).

Studium stability nelineárních dyn. systémů je obtížné právě proto, že neexistuje jednoznačný vztah mezi stabilitou kritického bodu nelineárního systému a stabilitou téhož kritického bodu v odpovídajícím linearizovaném systému. V některých případech je však možno na základě analýzy linearizovaného systému učinit cenné závěry o chování systému nelineárního. Pro dvoudimenzionální dynamické systémy platí tato věta [43].

Věta 10 *Nechť je dán dynamický systém s pohybovými rovnicemi*

$$\frac{dx_1}{dt} = a_{11}x_1 + a_1 2x_2 + \Phi(x_1, x_2), \frac{dx_2}{dt} = a_2 1x_1 + a_2 2x_2 + \Psi(x_1, x_2). \qquad (5.5)$$

Nechť jsou dále splněny podmínky:

1. $\Phi(0, 0) = \Psi(0, 0) = 0$ *a okolí počátku souřadnic neobsahuje žádné jiné kritické body;*

2. $$\lim_{x_1, x_2 \to 0} \frac{\Phi(x_1, x_2)}{\sqrt{(x_1^2 + x_2^2)}} = \lim_{x_1, x_2 \to 0} \frac{\Psi(x_1, x_2)}{\sqrt{(x_1^2 + x_2^2)}} = 0;$$

3. *vlastní čísla* λ_1, λ_2 *matice* $\mathbf{A}$ *jsou obě nenulová a mají také nenulové reálné části.*

Pak je chování nelineárního systému s pohybovými rovnicemi (5.5) v okolí počátku stejné jako chování příslušného linearizovaného systému.

Důkaz tohoto tvrzení je uveden v monografii [43]. Je třeba zdůraznit, že linearizace je validní pouze v dostatečně malém okolí počátku souřadnic.

Z uvedených vět vyplývá kritérium pro posouzení stability daného kritického bodu obecného dynamického systému s pohybovými rovnicemi (5.1). O stabilitě kritického bodu $[c_1, c_2 \ldots, c_n]$ rozhodují vlastní čísla matice $\mathbf{D}$,

$$d_{ij} = \frac{\partial f_i(c_1, c_2, \ldots, c_n)}{\partial x_j}; \quad i, j = 1, 2, \ldots, n.$$

Jednoznačný závěr lze učinit pouze v případě tzv. hyperbolického kritického bodu, kdy žádné vlastní číslo příslušné matice $\mathbf{D}$ nemá nulovou reálnou část. **Hyperbolický kritický bod** je:

1. asymptoticky stabilním uzlem nebo ohniskem, jestliže všechna vlastní čísla matice $\mathbf{D}$ mají zápornou reálnou část;

2. nestabilním uzlem nebo ohniskem, jestliže všechna vlastní čísla matice $\mathbf{D}$ mají kladnou reálnou část;

3. nestabilním sedlem, jestliže některá vlastní čísla matice $\mathbf{D}$ mají kladnou a jiná zápornou reálnou část.

Závěrem se ještě stručně zmíníme o stabilitě uzavřených (periodických) trajektorií. Uvažujme dynamický systém s pohybovými rovnicemi (5.1) a předpokládejme, že jsou splněny předpoklady pro existenci a jednoznačnost jeho řešení. Dále předpokládejme, že $\gamma(t)$ je nějaká uzavřená trajektorie uvažovaného systému s periodou $\tau > 0$. Pak platí následující tvrzení (viz [34, 43]):

1. Uzavřená trajektorie takového systému musí obsahovat aspoň jeden kritický bod ležící uvnitř této trajektorie. Jestliže obsahuje právě jeden kritický bod, pak je tímto bodem buď uzel nebo ohnisko nebo střed.

2. Nutnou podmínkou pro existenci uzavřené trajektorie v oblasti A dvoudimenzionálního systému je, aby funkce

$$J(x_1, x_2) = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2}$$

byla buď identicky nulová v oblasti A nebo měnila znaménko v oblasti A (**Bendixonovo negativní kritérium**).

3. Kritériem stability uzavřené trajektorie $\gamma(t)$ jsou vlastní čísla tzv. **matice monodromie $\boldsymbol{\Delta}$**,

$$\Delta_{ij} = \frac{\partial \phi_i}{\partial x_j}; \quad i, j = 1, 2, \ldots, n,$$

kde $\phi : \mathrm{R} \times \mathrm{R}^n \longrightarrow \mathrm{R}^n$ je zobrazení takové, že pro každý bod $\mathbf{x} \in \gamma(t)$ platí $\phi(\tau, \mathbf{x}) = \mathbf{x}$.

Vlastní čísla matice $\boldsymbol{\Delta}$ nezávisejí na volbě bodu $\mathbf{x} \in \gamma(t)$. Tato matice má jeden vlastní vektor tečný k uzavřené trajektorii $\gamma(t)$ a jemu přísluší vlastní číslo 1. Jsou-li ostatní vlastní čísla matice $\boldsymbol{\Delta}$ v absolutní hodnotě menší než 1, pak je uzavřená trajektorie $\gamma(t)$ asymptoticky stabilní.

5.2 Metody numerické integrace

Abstraktním modelem spojitého dynamického systému je obecně soustava diferenciálních a algebraických rovnic, proto je třeba věnovat zvláštní pozornost numerickým metodám pro jejich řešení.

Numerické řešení algebraických ani transcendentních rovnic nepředstavuje zpravidla vážnější problém, pokud máme k dispozici vhodnou univerzální metodu, např. Newtonovu metodu. Naproti tomu však různorodost modelovaných systémů vyžaduje, aby byly programové prostředky pro spojitou simulaci vybaveny řadou univerzálních i speciálních metod pro numerickou integraci.

V této části se budeme podrobněji zabývat pouze numerickými metodami řešení obyčejných diferenciálních rovnic a jejich soustav. Základní informace nalezne čtenář ve skriptech [41], podrobnější poučení v monografiích [37] a [42].

5.2.1 Základní pojmy

Uvažujme pro jednoduchost soustavu n obyčejných diferenciálních rovnic 1. řádu

$$\frac{dy_i}{dt} = f_i(t, y_1, y_2, \ldots, x_n) \tag{5.6}$$

s počátečními podmínkami $y_i = y_{i0}$, kde $i = 1, 2, \ldots, n$.

Tuto počáteční (Cauchyho) úlohu budeme zapisovat v maticovém tvaru

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \tag{5.7}$$

s počáteční podmínkou $\mathbf{y}(t_0) = \mathbf{y}_0$. Dále předpokládejme, že jsou splněny podmínky existence a jednoznačnosti řešení v oblasti $\Omega \times I$, kde Ω je stavový prostor uvažovaného systému a I nějaký interval reálných čísel.

Numerickým řešením počáteční úlohy se rozumí posloupnost $\{y_i\}$ hodnot

$$y_0 = y(t_0), y_1 = y(t_1), \ldots, y_k = y(t_k),$$

které odpovídají hodnotám t_i, $i = 1, 2, \ldots, k$, nezávisle proměnné t (zpravidla času) v intervalu $< t_0, t_k >$. Hodnota výrazu $h = t_{i+1} - t_i$ přitom udává velikost **integračního kroku**. **Exaktní řešení** uvažované počáteční úlohy budeme značit $Y_i = Y(t_i)$, $i = 1, 2, \ldots, k$.

Cílem numerických metod je nalezení numerického řešení. Přitom se požaduje, aby posloupnost $\{y_i\}$ konvergovala pro $h \to 0$ k exaktnímu řešení $Y(t_i)$.

Rozlišujeme dva základní typy metod řešení soustavy (5.7):

- Metody, kde se hodnoty funkce $\mathbf{f}(t, \mathbf{y})$ počítají jen v bodech $[t_i, \mathbf{y}_i]$, přičemž $\mathbf{y}_i$ je hodnota numerického řešení v bodě $t = t_i$. Do této skupiny patří především tzv. **víceuzlové metody**.

- Metody, kde se hodnoty funkce $\mathbf{f}(t, \mathbf{y})$ počítají navíc v bodech ležících mezi jednotlivými body $[t_i, \mathbf{y}_i]$, $i = 1, 2, \ldots, k$. Zástupci těchto metod jsou především **metody Rungeho-Kutty**.

V obou případech je posloupnost hodnot y_i výsledkem postupné extrapolace, přičemž již výchozí hodnoty v každém kroku jsou zatížený **lokální chybou**.

Tato chyba je součtem dvou příspěvků: **chyby metody** (**truncation error**), která je způsobena respektováním jen konečného počtu členů Taylorova rozvoje funkce **f**, a **zaokrouhlovací chyby** (**rounding error**), jež souvisí s omezenou délkou slova v paměti počítače. Chyba jednoho kroku pak přirozeně ovlivňuje výsledky kroků následujících. Celková chyba ε po realizaci n kroků, tzv. **akumulovaná chyba po n krocích**, je dána vztahem

$$\varepsilon = Y_n - y_n.$$

Kvalita použité numerické metody se hodnotí podle těchto základních kritérií:

- přesnost metody (velikost lokální chyby a prostředky pro její odhad),

- stabilita metody,

- časová náročnost výpočtu,

- nároky na operační paměť počítače.

Poznámka.
Problematika stability numerického řešení soustavy (5.7) přesahuje rámec těchto skript. Zavedeme proto jen pojem absolutní stability. Metoda je **absolutně stabilní** pro daný krok h a danou soustavu diferenciálních rovnic, jestliže

chyba vzniklá při výpočtu hodnoty $\mathbf{y}_n$ se nezvětšuje při výpočtu následujících hodnot $\mathbf{y}_k$, $k > n$. K vyšetřování absolutní stability se používá testovací rovnice $y' = \lambda y$, kde λ je konstanta (obecně komplexní číslo). Množina hodnot $\tilde{h} = h\lambda$, pro něž je metoda absolutně stabilní, se nazývá **obor absolutní stability** příslušné soustavy.

5.2.2 Metody Rungeho-Kutty

Těmto metodám se také říká **jednouzlové**, protože k výpočtu hodnoty $\mathbf{y}_{n+1}$ stačí znát pouze hodnotu y_n v bezprostředně předcházejícím uzlu.

Vychází se obecně ze vztahu

$$\mathbf{y}_{n+1} - \mathbf{y}_n = \sum_{i=1}^{p} w_i \mathbf{k}_i, \tag{5.8}$$

kde w_i jsou konstanty a

$$\mathbf{k}_i = h\mathbf{f}(t_n + a_i h, \mathbf{y}_n + \sum_{j=1}^{i-1} b_{ij}\mathbf{k}_j)$$

pro $i = 1, 2, \ldots, p$, $h = t_{n+1} - t_n$, a_i a b_{ij} jsou konstanty, přičemž $a_1 = 0$. Metoda popsaná vztahem (5.8) se nazývá **p-hodnotová**, protože používá právě p hodnot funkce $\mathbf{f}(t, \mathbf{x})$. Konstanty w_i, a_i, b_{ij} se spočtou tak, aby získané řešení souhlasilo s Taylorovým rozvojem v bodě $[t_n, \mathbf{y}_n]$ až do P-té mocniny integračního kroku h včetně. Taková metoda se pak nazývá **metoda Rungeho-Kutty řádu** P. Pro $p \leq 4$ zřejmě platí $P = p$.

Příklady metod Rungeho-Kutty:

1. Metoda 1. řádu (Eulerova)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$$

2. Příklad metody 2. řádu

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}(\mathbf{f}_n + \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{f}_n))$$

   Je-li $\mathbf{f}(t, \mathbf{y})$ pouze funkcí času t, jde o aplikaci lichoběžníkového pravidla.

3. Příklad metody 3. řádu

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{1}{6}(\mathbf{k}_1 + 4\mathbf{k}_2 + \mathbf{k}_3),$$

kde

$$\begin{aligned}\mathbf{k}_1 &= h\mathbf{f}(t_n, \mathbf{y}_n),\\ \mathbf{k}_2 &= h\mathbf{f}(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{\mathbf{k}_1}{2}),\\ \mathbf{k}_3 &= h\mathbf{f}(t_n + h, \mathbf{y}_n - \mathbf{k}_1 + 2\mathbf{k}_2).\end{aligned}$$

Je-li $\mathbf{f}(t, \mathbf{y})$ pouze funkcí času t, jde v podstatě o použití Simpsonova pravidla.

4. Metody 4. řádu mají obecný tvar

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \sum_{i=1}^{4} w_i \mathbf{k}_i,$$

kde

$$\begin{aligned}\mathbf{k}_1 &= h\mathbf{f}(t_n, \mathbf{y}_n),\\ \mathbf{k}_2 &= h\mathbf{f}(t_n + a_2 h, \mathbf{y}_n + b_{21}\mathbf{k}_1),\\ \mathbf{k}_3 &= h\mathbf{f}(t_n + a_3 h, \mathbf{y}_n + b_{31}\mathbf{k}_1 + b_{32}\mathbf{k}_2),\\ \mathbf{k}_4 &= h\mathbf{f}(t_n + a_4 h, \mathbf{y}_n + b_{41}\mathbf{k}_1 + b_{42}\mathbf{k}_2 + b_{43}\mathbf{k}_3).\end{aligned}$$

Jednotlivé metody 4. řádu se odlišují volbou konstant $w_i, a_i, b_{ij},\ i, j = 1, 2, 3, 4$.

K odhadování přesnosti metod Rungeho-Kutty se používá tzv. metoda polovičního kroku, tj. dvojí výpočet: jednak s krokem h, jednak s krokem $2h$. Všechny metody Rungeho-Kutty mají ohraničený obor absolutní stability; jeho rozsah se zvětšuje s rostoucím řádem metody. Jejich implementace na počítači je velmi jednoduchá, integrační krok lze libovolně měnit. Mají přibližně stejnou, často i vyšší přesnost než metody prediktor-korektor stejného řádu (viz odstavec 5.2.3).

5.2.3 Víceuzlové metody

Numerická metoda se nazývá k**-uzlová**, jestliže k výpočtu hodnoty $\mathbf{y}_{n+1}$ používá právě k aproximací bezprostředně předcházejících. Vychází se z obecného vztahu

$$\mathbf{y}_{n+1} = \sum_{i=0}^{r} a_i \mathbf{y}_{n-i} + h \sum_{i=-1}^{r} b_i \mathbf{y'}_{n-i}, \tag{5.9}$$

kde a_i, b_i jsou konstanty. Ve vztahu (5.9) platí $k = r + 1$.

Uvedené metody nejsou samostartující. Nejprve je nutno spočítat prvních k hodnot některou z metod typu Rungeho-Kutty nebo jinou jednouzlovou metodou.

Metoda definovaná vztahem (5.9) je řádu právě p, jestliže je přesná pro polynomy stupně p. Přitom se vždy požaduje $p \geq 2$. Některé z koeficientů a_i, b_i mohou být rovny nule.

V podstatě se rozlišují tři základní skupiny víceuzlových metod.

1. Je-li $b_{-1} = 0$, pak hodnota $\mathbf{y}_{n+1}$ je vyjádřena jako lineární kombinace již známých aproximací $\mathbf{y}_i$. Takové metody se nazývají **explicitní** (**přímé** nebo **prediktorového typu**).

2. Je-li ovšem $b_{-1} \neq 0$, pak (5.9) představuje implicitní rovnici pro $\mathbf{y}_{n+1}$, protože $\mathbf{y'}_{n+1} = \mathbf{f}(t_n + 1, \mathbf{y}_{n+1})$, a takovou rovnici lze řešit jen iteračně. Metody této skupiny se nazývají **implicitní** (**nepřímé**, **iterační** nebo **korektorového typu**).

3. Kombinací obou předchozích typů vznikají **metody prediktor - korektor**. Jejich princip je jednoduchý. Pro první výpočet (predikci) $\mathbf{y}^P_{n+1}$ se používá formule explicitní. Následně se určí derivace $\mathbf{y'}_{n+1} = \mathbf{f}(t_{n+1}, \mathbf{y}^P_{n+1})$ a pak se hledané řešení zpřesňuje (koriguje) pomocí implicitní formule (počítá se $\mathbf{y}^C_{n+1}$). Dvojice metod prediktor-korektor se vybírá tak, aby obě vybrané metody měly stejný řád.

Z velkého počtu víceuzlových metod (viz např. [37] nebo [42]) uvedeme jen několik málo příkladů. Přitom zavedeme označení $\mathbf{y'}_i = \mathbf{f}_i$ pro všechna i.

1. Adamsovy (Bashforthovy) prediktory

$$\begin{aligned}
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}(3\mathbf{f}_n - \mathbf{f}_{n-1}), \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{12}(23\mathbf{f}_n - 16\mathbf{f}_{n-1} + 5\mathbf{f}_{n-2}), \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{24}(55\mathbf{f}_n - 59\mathbf{f}_{n-1} + 37\mathbf{f}_{n-2} - 9\mathbf{f}_{n-3}).
\end{aligned}$$

2. Adamsovy korektory

$$\begin{aligned}
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{12}(5\mathbf{f}_{n+1} + 8\mathbf{f}_n - \mathbf{f}_{n-1}), \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{24}(9\mathbf{f}_{n+1} + 19\mathbf{f}_n - 5\mathbf{f}_{n-1} + \mathbf{f}_{n-2}).
\end{aligned}$$

Implicitní metody jsou sice pracnější než explicitní, ale jsou obecně přesnější a mají lepší vlastnosti, pokud jde o numerickou stabilitu.

5.2.4 Metody pro řešení tuhých (stiff) soustav

Soustava obyčejných diferenciálních rovnic typu (5.7) se nazývá **tuhou** (anglicky **stiff**) **soustavou**, jestliže vlastní čísla λ_i její Jacobiho matice

$$\mathbf{J} = \left(\frac{\partial f_i}{\partial x_j} \right), \qquad i, j = 1, 2, \ldots, n,$$

jsou značně rozdílná. Je zřejmé, že vlastní čísla matice **J** závisejí na čase a v průběhu integrace se tedy mění.

Tuhá soustava obyčejných diferenciálních rovnic má tyto vlastnosti:

- $\mathrm{Re}\lambda_j < 0$, $j = 1, 2, \ldots, n$, pro všechny hodnoty t,
- poměr

$$R = \frac{\max\limits_j \mid \lambda_j \mid}{\min\limits_j \mid \lambda_j \mid}$$

  je velké číslo (v typických úlohách z praxe řádu 10^3 až 10^6).

Poměr R je charakteristikou soustavy obyčejných diferenciálních rovnic typu (5.7); nazývá se jejím **tuhostním poměrem**.

U většiny numerických metod stabilita metody vyžaduje omezení integračního kroku h ve tvaru $\mid h\lambda \mid < K$, kde K je konstanta charakteristická pro danou metodu a λ je v absolutní hodnotě největší vlastní číslo Jacobiho matice. Pro velké hodnoty λ je tedy nutno volit malý integrační krok.

K řešení tuhých soustav jsou tedy vhodné metody, jejichž oblast absolutní stability je celá levá polorovina, tj. $\mathrm{Re}\lambda h < 0$. Takové metody se nazývají **A-stabilní**. Pro tyto metody platí:

- Explicitní víceuzlové metody nemohou být A-stabilní.
- A-stabilní implicitní metody mohou být nejvýše 2. řádu.

Příklady A-stabilních metod:

1. implicitní metoda 1. řádu (Eulerova metoda)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}_{n+1},$$

2. implicitní metoda 2. řádu (lichoběžníková metoda)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}(\mathbf{f}_{n+1} + \mathbf{f}_n).$$

Požadavek A-stability je příliš silný, neumožňuje používat víceuzlových metod řádu vyššího než 2. Proto byly zavedeny tzv. **tuhostně stabilní metody**. Základní poučení o těchto metodách nalezne čtenář např. ve skriptech [41].

5.2.5 Posouzení metod numerického řešení soustav obyčejných diferenciálních rovnic

Na závěr uvedeme několik poznámek k hodnocení jednotlivých numerických metod podle kritérií zavedených v odstavci 5.2.1.

Přesnost metody. Metody vyšších řádů jsou obecně přesnější (mají menší lokální chybu) než metody nižších řádů téhož typu. U víceuzlových metod jsou metody implicitní přesnější než explicitní.

Stabilita metody. Pro metody Rungeho-Kutty platí, že s rostoucím řádem metody se zvětšuje oblast její absolutní stability. Naproti tomu u Adamsových explicitních metod a metod prediktor - korektor se oblast absolutní stability zmenšuje s rostoucím řádem metody.

Časová náročnost výpočtu. Rychlost řešení je pro danou hodnotu integračního kroku h nepřímo úměrná řádu metody. Víceuzlové metody jsou při daném h obecně rychlejší než metody jednouzlové.

Nároky na paměť. Tyto nároky jsou obecně přímo úměrné řádu metody. Pro metody téhož řádu platí, že víceuzlové metody jsou náročnější než metody jednouzlové. U víceuzlových metod mají implicitní metody větší nároky na paměť než metody explicitní.

Kapitola 6

Příklady na simulaci

6.1 Modely hromadné obsluhy

6.1.1 Úvod

Systémy hromadné obsluhy se anglicky nazývají **queuing systems**, což můžeme volně přeložit jako systémy, v nichž jsou fronty. Fronty, tj. uspořádané seznamy s režimem FIFO (first in, first out), jsou nejdůležitějšími prvky takových systémů. Tvoří většinou pevné struktury (tak zvané **báze**) spolu s jinými prvky, kvůli nimž tyto fronty vznikají. Jde o prvky, které se nazývají **facility** (podle anglického termínu **facility**) a **sklady** (podle anglických termínů **storage** nebo **store**). Po této pevné struktuře se pohybují **transakce**.

Facilita je prvek, který je v každém okamžiku schopen interagovat nejvýše s jednou transakcí (v profesi systémů hromadné obsluhy se častěji říká obsloužit nejvýš jednu transakci); takové obsloužení trvá nějakou dobu a během ní ovšem může požadovat obsloužení jiná transakce. Když k tomu dojde, musí tato transakce čekat ve frontě, až se facilita uvolní a až tato transakce přijde na řadu. Sklad je podobný facilitě, avšak je schopen najednou obsloužit více transakcí, maximálně jistý počet, který se nazývá **kapacitou skladu**. Je sice evidentní, že facilita je sklad s kapacitou rovnou jedné, avšak v praxi se mezi facilitami a sklady rozlišuje: facilita může být buď volná nebo obsazená, kdežto sklad může být také „částečně obsazený“.

Kromě facilit a skladů mohou být v bázích i jiné prvky, např. brána (anglicky **gate**). Je to jakési zobecnění výtahu obsluhovaného lidským operátorem: má jistou kapacitu a začne obsluhovat všechny transakce, teprve když je počet transakcí roven této kapacitě. Zobecnění spočívá v tom, že „gate“ je obecně jakási brána (anglicky gate), před kterou jakoby čekaly prvky systému, jež na ni narazí při provádění svých životních pravidel, a jsou do ní (a dále za ni) vpuštěny, když je splněna jistá podmínka. Avšak takové prvky se nevyskytují

$$G_1 \to Q_1 \to Z_1 \to S_1 \to Q_3 \to Z_2 \to Q_4 \to \begin{array}{c} S_2 \\ \uparrow \\ Q_6 \\ \uparrow \\ G_2 \end{array} \to Q_5 \to Z_3 \to$$

Obrázek 6.1: Ilustrace k příkladu na použití jazyka GPSS

často, a tak se v tomto výkladu omezíme jen na transakce, facility, sklady a fronty. V obecném případě má každá transakce atribut zvaný **priorita** a v závislosti na ní se mohou transakce ve frontách předbíhat, avšak i tento rys systémů hromadné obsluhy budeme v dalším výkladu pro jednoduchost ignorovat.

Prvky báze systému jsou spojeny „cestami“ transakcí. V profesi systémů hromadné obsluhy se vztah mezi frontou a tím prvkem, kvůli němuž existuje, vyjadřuje předložkou „před“. Říká-li se, že fronta je před facilitou, resp. skladem, znamená to, že z takové fronty vede k příslušné facilitě či skladu přímá cesta, a když je tento prvek (facilita či sklad) obsazen, čeká transakce, která s ním má interagovat, v té frontě. Mezi prvky každé báze je třeba zahrnout také jeden nebo několik **generátorů** transakcí, které reprezentují vstupy transakcí do systému. Takové generátory nejčastěji produkují transakce nezávisle na stavu systému, avšak v některých případech takto reagují na vstup transakce: produkují jakousi její „dceru“ a mohou tedy reprezentovat místa, kde se transakce dělí. V bázi je také jeden nebo více **spotřebitelů** (anglicky **sink**, tedy přesněji odpad, výtok, díra), což jsou prvky, v nichž transakce mizí (čili opouštějí systém). Občas se v bázi vyskytují i prvky, které transakce slučují, tj. několik transakcí nechají zmizet a místo nich generují transakci jednu, jejíž některé atributy jsou kopiemi transakcí, které do takového prvku vstoupily. Ve výrobním procesu si můžeme takto představit montážní krok, to jest spojení několik součástek na jeden produkt, který dále už vystupuje jako jeden celek. I takové prvky, kterými jsou transakce slučovány, budeme dále v našem výkladu ignorovat.

6.1.2 Příklad modelu systému hromadné obsluhy v jazyku GPSS

Uvažujme systém hromadné obsluhy znázorněný na obr. 6.1, kde Q_i značí frontu, Z_i facilitu, S_i sklad a G_i vstup do systému. Předpokládejme pro jednoduchost, že generátor G_1 posílá do systému jednu transakci přesně každou desátou minutu a že doby mezi dvěma následujícími vstupy transakcí generátorem G_2 mají normální rozdělení se střední hodnotou pět minut a směrodatnou

odchylkou 3 minuty, avšak že první transakce vstoupí prvkem G_2 až po 30. minutě. Předpokládejme dále, že doby obsluhy mají také normální rozdělení, a to ve facilitách Z_1, Z_2 a Z_3 po řadě $N(6,3), N(5,2)$ a $N(4,2)$ a ve skladech S_1 a S_2 po řadě $N(40,15)$ a $N(50,20)$. Kapacita S_1 nechť je 10 a kapacita S_2 nechť je 12, po zpracování ve facilitě Z_3 nechť transakce opustí systém a nechť je simulační pokus ukončen, když generátorem G_1 vstoupí do systému tisíc transakcí a generátorem G_2 šest set transakcí.

První implementace amerického jazyka GPSS (General Purpose Systems Simulator) [44, 22], zaměřeného na simulaci systémů hromadné obsluhy, se datuje od roku 1961, takže jde o první jazyk pro diskrétní simulaci na světě.

Ukážeme, jak se v něm uvedený systém popíše.

```
S1 STORAGE   10
S2 STORAGE   12
   GENERATE  10,,0,1000
   SEIZE     Z1
   ADVANCE   6,3
   RELEASE   Z1
   ENTER     S1
   ADVANCE   40,15
   LEAVE     S1
   SEIZE     Z2
   ADVANCE   5,2
   RELEASE   Z2
   ENTER     S2
   ADVANCE   50,20
   LEAVE     S2
   SEIZE     Z3
   ADVANCE   4,2
   RELEASE   Z3
   TERMINATE
   GENERATE  5,3,30,600
   ENTER     S2
   ADVANCE   50,20
   LEAVE     S2
   SEIZE     Z3
   ADVANCE   4,2
   RELEASE   Z3
   TERMINATE
```

Vidíme, že jazyk GPSS vede svého uživatele k tomu, že rozpoznává dva různé „druhy životů“ transakce, vlastně dvě různé třídy transakcí, které jsou zají-

mavé tím, že společně žádají o obsluhu jisté facility a sklady. Facilitu nemusí uživatel vůbec speciálně zavádět, stačí, když její jméno uvede alespoň jednou v životních pravidlech. Sklady zavést musí, a to kvůli určení jejich kapacity. Životní pravidlo `SEIZE X` stanoví, že transakce požádá facilitu `X` o obsluhu - pokud je tato facilita obsazena, čeká transakce automaticky ve frontě „před“ ní. Životní pravidlo `ADVANCE A,B` stanoví, že transakce počká, až se hodnota simulovaného času zvýší o náhodnou veličinu, která má normální rozdělení se střední hodnotou A a směrodatnou odchylkou B. Životní pravidlo `RELEASE X` stanoví, že transakce uvolní facilitu `X`. Trojice právě popsaných životních pravidel tedy vystihuje to, že transakce žádá danou facilitu o obsluhu, která má trvat po dobu určenou normálním rozdělením $N(A, B)$.

Dvojice `SEIZE` a `RELEASE` mají analogii `ENTER` (vstup) a `LEAVE` (odejdi) pro sklady. `TERMINATE` (ukonči) říká jednak to, že transakce má opustit systém, a jednak to, že končí popis životních pravidel.

Poměrně bohatý výběr možností dává životní pravidlo `GENERATE`. Uvozuje životní pravidla, je jistým obrazem toho, že transakce, která se podle nich chová, právě vychází z jistého vstupu, a dává tomuto vstupu jisté vlastnosti, a to pomocí argumentů, které za slovem `GENERATE` následují. První stanoví průměrnou dobu mezi dvěma následujícími příchody transakcí tímto generátorem; v druhém je zakódován rozptyl; třetí argument říká, kdy vypustí generátor první transakci, a čtvrtý stanoví, kolik transakcí do systému generátorem vstoupí. Po vstupu poslední transakce přestane generátor pracovat.

V popisu modelu se vůbec nevyskytují fronty Q_1 až Q_6: jazyk GPSS totiž automaticky modeluje frontu před každou facilitou a před každým skladem, takže v běžném případě je nemusíme vůbec zavádět. Totéž platí pro výstup dat - při ukončení simulačního pokusu vystoupí automaticky tabulky a histogramy charakterizující využití jednotlivých facilit a skladů.

6.1.3 Popis modelu v jazyku SIMULA

Naznačme, jak by se uvedený systém hromadné obsluhy popsal v jazyku SIMULA [45].

```
GPSS
begin integer U, pocet1, pocet2; ref(facility) Z1, Z2, Z3;
      ref(storage) S1,S2;
      process class generator1
      begin loop: activate new vyrobek1; hold(10);
            if pocet1 lt 1000 then go to loop
      end;
      process class generator2;
      begin loop: activate new vyrobek2; hold(normal(5,3,U));
```

```
            if pocet2 lt 600 then go to loop
      end;
      transaction class vyrobek1;
      begin pocet1:=pocet1+1;
            seize(Z1); hold(normal(6,3,U); release(Z1);
            enter(S1); hold(normal(40,15,U); leave(S1);
            seize(Z2); hold(normal(5,2,U); release(Z2);
            enter(S2); hold(normal(50,20,U); leave(S2);
            seize(Z3); hold(normal(4,2,U); release(Z3)
      end;
      transaction class vyrobek2;
      begin pocet2:=pocet2+1;
            enter(S2); hold(normal(50,20,U); leave(S2);
            seize(Z3); hold(normal(4,2,U); release(Z3)
      end;
      S1:-new storage(10); S2:-new storage(12);
      F1:-new facility; F2:-new facility; F3:-new facility;
      activate new generator1; activate new generator2
      delay 30;
      hold(100000); ...
end;
```

Generátory jsou popsány jako zvláštní třídy `generator1` a `generator2`. Od každé z nich je pak vytvořena jedna instance. Ta první je aktivována ihned: ihned tedy začne generovat transakce třídy `vyrobek1`; ta druhá je aktivována se **zpožděním** (anglicky **delay**) třiceti časových jednotek, takže začne generovat instance třídy `vyrobek2` až počínaje časem 30. Na počátku se též vytvoří potřebné facility a sklady a pak se čeká 100000 časových jednotek, během nichž pracují generátory a generované transakce. Předpokládá se, že vše skončí do oněch 100000 časových jednotek. Pak se nechají vystoupit výsledky, což je zde naznačeno třemi tečkami. Zcela na začátku je prefix „GPSS“. Tím jsme naznačili, že autor programu aplikuje definice, které byly shrnuty do třídy nazvané GPSS. SIMULA je totiž obecně použitelný (univerzální, tedy nejen simulační) jazyk, takové věci jako transaction, storage, facility, seize, release, enter a depart jsou pro něj příliš specializované, takže je třeba je naprogramovat. Předpokládá se, že někdo v něm definoval prostředky jazyka GPSS, aniž by pro tento jazyk pracně implementoval kompilátor. Celočíselná proměnná `U` je zde rezervována pro generátor pseudonáhodných čísel.

6.2 Kompartmentové modely

6.2.1 Co jsou kompartmentové systémy?

Všichni víme, že hmota je složena z molekul, ty z atomů, ty z jádra a elektronového obalu atd., avšak všichni máme zkušenost, že na příklad v běžné fyzice se toto složení hmoty ignoruje a hmotu chápeme jako cosi spojitého. Tento přístup je vlastní mnoha technickým aplikacím v chemii (např. stechiometrické výpočty) a mnoha aplikacím, které mají k chemii blízko - konkrétně jde o biomedicínský přístup k organismům nebo orgánům. Pro řadu aplikací je totiž výhodné chápat zkoumaný systém jako složený ze „zásob“ látek, mezi nimiž protéká látka „kanály“. Zásoba látky může reprezentovat jisté umístění (na příklad krev v plicích, v srdci nebo v játrech, voda v moři, v mracích nebo v řekách), nebo jistý chemický stav (surová síra nebo síra vázaná ve vyrobené kyselině sírové nebo síra vázaná v oxidu siřičitém), nebo jistý biologický stav (což může někdy být velmi blízko chemickému stavu - např. železo v kostní dřeni, v erytrocytech, v krevní plasmě nebo v potravě, jindy však dosti daleko - např. dospělý hmyz, hmyz zakuklený, larvy nebo vajíčka, nebo děti předškolního věku, děti školou povinné, učni, středoškoláci, vysokoškoláci, zaměstnanci, nezaměstnaní, podnikatelé, či penzisté).

Vztahy mezi jednotlivými molekulami, atomy či jinými „individui“ jsou velmi složité, avšak často se naše poznání zjednoduší, když velká množství jednotlivých individuí nahradíme homogenními zásobami spojenými kanály, jejichž objem je sice nulový, ale jimiž přesto v kladném čase může projít z jedné zásoby do jiné nenulové množství „látky“ složené ze zmíněných individuí. Idealizované jsou ovšem nejen kanály, ale i samy zásoby. Slovo homogenní je interpretováno tak, jako by každé individuum, které do zásoby vstoupí, bylo v tomtéž stavu (místě atd.) jako všechna ostatní individua, jež v uvažované zásobě byla už dříve. Proto se říká, že zásoby jsou „dobře promíchány“ (v anglické literatuře **well mixed** nebo **well stirred**). A tuto abstrakci dělají různé profese medicíny, ekologie i ekonomie. Místo slova zásoba se používá odborný termín **kompartment** (anglicky **compartment**). Systém složený z kompartmentů spojených idealizovanými kanály se nazývá **kompartmentovým systémem** (anglicky **compartmental system**).

Kompartmentové systémy bývají často chápány i tak, že „látka“, která jimi prochází, má cosi jako svůj objem a případně nějakou další vlastnost, která podléhá míchání v kompartmentech. V nukleární medicíně, kde precizní studium kompartmentových systémů proběhlo nejdříve, byla tato vlastnost radioaktivní stopovací látkou, a tak se pro takovou vlastnost ustálil anglický termín **tracer**. Když se na příklad sleduje, proč někomu nepracuje ledvina tak, jak se u zdravého člověka předpokládá, vpraví se mu do krve neškodné množství

radioaktivního jodu a ten funguje jako stopovací látka. Mimo tělo takto sledovaného člověka lze měřit radioaktivní záření jodu, který je v jedné či druhé ledvině, v močovém měchýři, v srdci, v odebrané dávce krve atd., a podle toho, jak se úroveň radioaktivity mění během jisté doby od jejího vpravení do organismu, lze stanovit diagnózu: ledvina je např. zmenšená nebo rychlost její činnosti je zpomalená apod.

Místo radioaktivního prvku může ovšem vystupovat např. lék, který je v organismu transportován, nebo látka, jež do systému vlastně nepatří (např. katabolity zhoubného nádoru). Kompartmentový systém nemusí zobrazovat jen živý organismus. „Tracer“ může být např. znečištění složek životního prostředí nebo může být cosi zcela odlišného. Výstižným příkladem je cena za objemovou jednotku materiálu v případě, že kompartmentový systém zobrazuje nějaký výrobní proces: cena za jednotlivé složky (suroviny, meziprodukty, ...) se při montáži míchá, „přimíchá“ se k ní i cena za práci a za energii a v kompartmentu konečných výrobků představuje takto dosažený „tracer“ předpokládanou cenu za výrobek.

6.2.2 Diskrétní pojetí kompartmentových systémů

Chování kompartmentových systémů lze popsat obyčejnými diferenciálními rovnicemi, někdy dokonce rovnicemi s konstantními koeficienty (pokud se rychlost přesunu látky v kanálech v čase nemění). Kompartmentové systémy však jsou tak názorné, že je používají i profese, jejichž pracovníci mají ve velké většině k diferenciálním rovnicím dost daleko; např. pro lékaře nejrůznějších zaměření, ekology a ekonomy by diferenciální rovnice znamenaly právě negaci té názornosti.

A tak pro tyto a podobné profese se derivace, která zde vlastně reprezentuje okamžitou rychlost změny, nahrazuje změnou za nějaký časový interval konečné, od nuly dostatečně odlišné délky. Takto se derivace zamění diferencí a z diferenciálních rovnic vzniknou rovnice diferenční. Avšak i takové rovnice mohou výše zmíněné profese oželet. Když ekolog přemýšlí o tom, jak nově postavená přehrada změní vodní (a tím i biologický) režim v krajině za dvacet let, nezakládá své úvahy na rytmu nějakých sekund či mikrosekund, nýbrž na ročním rytmu, neboť ví, že všechna data pro jeho kompartmentový systém jsou jisté roční průměry, v nichž je ukryto kolísání vodních toků, vypařování a dalších faktorů během roku. Podobný rytmus platí i pro ekonomy, sociology a demografy, kteří hledají na příklad dlouhodobé prognózy makroekonomických změn, zaměstnanosti obyvatelstva, organizace školství, sportu či zdravotnické péče, nebo věkového rozložení obyvatelstva. Lékařům, kteří sledují průběh radioaktivní stopovací látky v organismu, stačí většinou chápat jen denní změny, pokud např. vědí, že některé pro ně důležité procesy probíhají v organismu ve

spánku jinak než při bdění.

Nezapomeňme, že systém je pouhá abstrakce reality. První krok abstrakce v oblasti kompartmentových systémů spočívá v tom, že se jednotlivá individua zanedbají a nastoupí spojitě se chovající látka. Druhý krok spočívá v tom, že kompartmentový systém existuje jakoby jen v konečném počtu od sebe (o rok, o den apod.) vzájemně vzdálených okamžicích a vždy právě v těchto okamžicích se určitá kladná množství látky přesunou mezi kompartmenty, načež se tam, kam se přesunula, ihned dobře promíchají s látkou, která tam zůstala.

Tato formulace zní jako přitažená za vlasy. Žádný ekolog, lékař či ekonom by takto studovaný systém nepopsal. My jsme ji však uvedli, a to ze dvou důvodů.

1. Předně jako příklad důsledně formulované systémové abstrakce, tedy formulace, která vystihuje to, co lékař, ekolog či ekonom chápe nějak mlhavě a nepřesně a přitom tuší, že "je to ono", o co mu jde;

2. Takto přesně formulovaný přístup ke kompartmentovým systémům je základem pojetí, že to jsou vskutku diskrétní systémy, ne nějaké nedokonale chápané systémy spojité.

6.2.3 Jazyk pro simulaci kompartmentových systémů

Popsané chápání kompartmentových systémů může sloužit za základ komunikace mezi člověkem a počítačem, tedy za základ simulačního jazyka, pomocí něhož uživatel-nematematik (lékař, ekolog,...) popíše to, co chce na počítači simulovat, zhruba tak, jako by to popsal svému kolegovi v profesi (tedy také nematematikovi).

Jazyků pro simulaci kompartmentových systémů bylo vytvořeno několik a liší se jednak svou „národností“ (různý přístup k realitě u různých národů), jednak svým „profesním zaměřením“. Známý americký jazyk DYNAMO byl původně navržen pro průmyslovou dynamiku (pro ekonomické modely dění v rámci oblastí odpovídajících našim okresům či krajům), pak jej jeho autor propagoval pro řízení světové průmyslové dynamiky (dění v rámci kontinentů), dále jej bezúspěšně nabízel pro tzv. komplexní dynamiku (v rámci jednotlivých výrobních podniků), až si ho všimli někteří ekologové. Je to tedy jazyk s širším použitím, má však nevýhodu, že má více variant, které se od sebe navzájem dosti liší. Považujeme tedy za rozumnější seznámit čtenáře s jiným jazykem, který je evropského původu a byl navržen ve své koncepci pro nukleární medicínu [21].

Nazývá se COSMO, což je zkratka slov **compartmental system modelling** (modelování kompartmentových systémů) a a vyšel přímo z praxe běžné na lékařských fakultách, na klinikách a v lékařských laboratořích. Pracovníci

těchto institucí si mezi sebou sdělují odborné informace o zkoumaných systémech tak, že jednotlivé kompartmenty graficky naznačí na papíře pomocí obdélníčků, jednotlivé kanály pomocí šipek mezi nimi, k tomu případně napíší vzorce, pokud se průtoková rychlost v kanálech mění, a pak do obdélníčků případně k šipkám napíší počáteční hodnoty systému, tj. objemy látky a množství (nebo koncentraci) stopovací látky pro jednotlivé kompartmenty a případně rychlosti přesunu.

Tento přístup lze aplikovat i v simulačním jazyku. Je třeba doplnit jen to, kdy má simulační pokus skončit, a to, co nás na modelu zajímá (tedy jeho výstupy).

Popis modelu se skládá z popisu jednotlivých kompartmentů (v nich jsou zahrnuty i popisy kanálů, jež do nich ústí), z popisu počátečního stavu, z formátu výstupu dat a z podmínky pro ukončení.

Kompartment je popsán v jediném „odstavci“. Ten začíná řádkou zvanou záhlaví. Její první slovo je `COMPARTMENT`, za ním je index popisovaného kompartmentu, tj. nějaké přirozené číslo, a pak název popisovaného kompartmentu, což může být jakýkoliv text. Pak následují řádky definující vstupy do popisovaného kompartmentu, tj. kanály, jimiž do kompartmentu vstupuje látka. Většinou vede do popisovaného kompartmentu kanál z jiného kompartmentu, kterému budeme říkat zdrojový. V tom případě je kanál popsán na jedné řádce, která začíná slovem `FROM`, za nímž následuje index zdrojového kompartmentu, pak rovnítko a za ním „výraz“ určující, kolik látky proteče za jeden simulační krok ze zdrojového kompartmentu do kompartmentu popisovaného. Slovo výraz jsme dali do uvozovek, protože až později popíšeme, co vše může být výrazem; zatím si lze představit nejjednodušší případ, totiž že za rovnítkem bude nějaká číselná konstanta. (V tom případě proteče daným kanálem v každém simulačním kroku konstantní množství látky.)

Uvedený výraz udává celkové množství látky, která daným kanálem proteče za jeden simulační krok. Simulační model z toho sám spočítá (na základě koncentrace stopovací látky ve zdrojovém kompartmentu), kolik stopovací látky projde kanálem v každém simulačním kroku.

Může se ovšem stát, že do kompartmentu vstupuje látka „zvenčí“, tj. že do kompartmentu ústí vstup do celého simulovaného systému. V tom případě nemá smysl mluvit o zdrojovém kompartmentu, a to se projevuje i v jazyku COSMO: místo `FROM` a následujícího indexu je v takovém případě třeba napsat slovo `PORTER`, za ním rovnítko a pak zase nějaký „výraz“ určující, kolik látky se do popisovaného kompartmentu dostává z okolí simulovaného systému v jednom časovém kroku. Slovo `PORTER` znamená nosič a v praxi kompartmentových systémů vyjadřuje toto slovo „čistou“ látku, tj. látku bez stopovací látky. Množství stopovací látky se nedá v tomto případě na základě nějaké koncentrace určit (zdrojový kompartment neexistuje). Pokud daným vstupem

přichází do simulovaného systému zvenčí i stopovací látka, musí být její množství, které tam přichází v simulačním kroku, určeno na dalším řádku, který začíná slovem `LABELLED`, za nímž je rovnítko následované opět nějakým „výrazem“. Slovo `LABELLED` opět vychází z praxe kompartmentových systémů, jak existuje v medicíně a ekologii; lze je přeložit jako „označená látka“ což vystihuje použití stopovací látky: např. železo v organismu může být „označeno“ radioaktivním izotopem železa, v jiných případech je krev „označena“ nějakým lékem, v ekologii může být např. stěhovavý pták označen kroužkem apod. V jiných případech může ovšem `PORTER` znamenat např. množství šroubků, které do systému odkudsi přicházejí, a `LABELLED` jejich úhrnnou cenu (tj. jakou cenu či hodnotu přicházející šroubky do simulovaného systému přinášejí). V takovém případě slovo `LABELLED` není příliš výstižné, ale nezbývá, než aby se obor, který používá kompartmentového pojetí jako pozdější (ekonomie) ve své komunikaci s počítačem přizpůsobil oboru (nukleární medicíně), který s kompartmentovým přístupem začal.

Popis jednoho kompartmentu tedy může vypadat např. takto

```
COMPARTMENT 1 ANORG. JODIDY
FROM2=0.5
FROM3=0.23
PORTER=0.2
LABELLED=30
```

a vyjadřuje fakt, že v simulovaném organismu je jod v anorganickém stavu chápán jako kompartment číslo 1 a má tři vstupy: z kompartmentu číslo 2 do něho přichází v každém simulačním kroku 0,5 (objemových) jednotek jodu, z kompartmentu číslo 3 do něho přichází 0,23 stejných jednotek jodu a zvenčí (např. potravou) do něho přichází 0,2 jednotek v každém kroku, které nesou množství 30 jednotek radioaktivního jodu. Čtenář se asi podiví, jak může 0,2 jednotek všeho přicházejícího jodu obsahovat 30 jednotek jodu radioaktivního. To není žádný nesmysl, neboť pro stopovací látku lze použít zcela jiných jednotek, nezávislých na rozhodnutí, v jakých jednotkách budeme vyjadřovat obecnou látku.

Přistupme nyní k vysvětlení toho, co znamená termín *výraz*. Každý kompartment je očíslován svým indexem. Je-li třeba, můžeme objem kompartmentu s indexem n identifikovat jako `V_n`, jeho vstup (totiž množství látky, vstupující do tohoto kompartmentu během daného časového kroku - tedy součet hodnot určených v těch řádcích popisu daného kompartmentu, které začínají slovy `FROM` nebo `PORTER}` jako `C_n` (`C` jako *come*) a jeho výstup (tj. součet hodnot určených všemi řádky, které začínají `FROMn`) jako `G_n` (`G` jako *go*). Uživatel dále může použít písmeno `H` před zmíněnými symboly, a tím uvede, že jde o množství stopovací látky: `HV3` je množství stopovací látky v kompartmentu

číslo 3, `HC1` je množství stopovací látky, která v daném simulačním kroku přichází do kompartmentu číslo 1, a `HG5` je množství stopovací látky, která v daném simulačním kroku opouští kompartment číslo 5.

Uživatel má možnost vytvářet výrazy z aritmetických operátorů (+, -, x, /), z celých čísel, z čísel s desetinnou tečkou a z identifikátorů tvaru `V_n`, `C_n`, `G_n`, `HV_n`, `HC_n` a `HG_n`. Nadto může používat okrouhlé závorky, některé matematické funkce jako `SIN`, `ABS`, `LN` apod. a identifikátor `STEP`, který značí délku simulačního kroku. Takže lze např. napsat

```
FROM4=ABS(324.65-V3)*STEP,
```

což znamená, že kanál, který vede do popisovaného kompartmentu z kompartmentu číslo 4, se bude tím více „otvírat“, čím více se bude odchylovat množství látky v kompartmentu číslo 3 od hodnoty 324,65, a dále že látky proteče kanálem tím více, čím delší je simulační krok, což má patrně význam v případě, kdy se délka simulačního kroku mění.

Jazyk COSMO má též odstavec pro určení počátečních hodnot. Ten začíná řádkou obsahující jediné slovo `INITIAL` a pak v každé další řádce je určena konvenčním způsobem jedna počáteční hodnota. Např. odstavec

```
INITIAL
V1=34
V2=27
V3=54
HV2=500
```

definuje, jaké budou na počátku simulačního pokusu objemy prvních tří kompartmentů a kolik bude stopovací látky ve druhém z nich. Pokud má být na začátku nějaká hodnota rovna nule, nemusí se v takovém odstavci vůbec definovat. V právě uvedeném příkladě by tedy v kompartmentech s indexy 1 a 3 nebyla na začátku žádná stopovací látka.

COSMO nabízí další prostředky, např. pomocné proměnné a jejich obhospodaření algoritmickým způsobem, prostředky pro vstup hodnot z klávesnice nebo jiného vstupního zařízení, prostředky pro grafický nebo tabulkový výstup informací o průběhu simulačního pokusu a pro podmínku, kdy má být simulační pokus ukončen. Jelikož se zběhlý uživatel jazyka COSMO může domnívat, že jej psaní delších slov zdržuje, nabízí COSMO možnost slova, která jsou v něm zavedena, zkracovat, takže odstavec pro popis kompartmentu anorganických jodidů uvedený výše může být zkrácen až na tento tvar:

```
C1
F2=0.5
F3=0.23
P=0.2
L=30
```

Simulační program v jazyku COSMO je tedy soubor odstavců respektujících výše formulovaná (a další podobná, zde už neuvedená) pravidla. Za posledním odstavcem je třeba napsat slovo `END` nebo alespoň `E`.

Na závěr uveďme, že dnes se simulují systémy o mnoha desítkách kompartmentů.

6.2.4 Buněčné systémy

6.2.4.1 Systémové rozdíly mezi kompartmentovými a buněčnými systémy

Kompartmentové systémy jsou v jistém smyslu velmi jednoduché. Každý z nich je tvořen pevným (na čase nezávislým) počtem prvků (kompartmentů a kanálů mezi nimi), a tyto prvky tvoří dokonce v čase neměnnou strukturu. Jediné, co se během času v kompartmentovém systému mění, jsou reálná čísla, tedy reálné atributy jeho prvků. Dalším znakem toho, jak je každý kompartmentový systém jednoduchý, je to, že změny ve všech jeho prvcích jsou synchronizovány. Systémová abstrakce chápe kompartmentové systémy tak, že se v nich ve spojitých časových intervalech nic neděje a jen občas, v konečně mnoha časových okamžicích, nastávají změny stavu kompartmentů, změny hodnot jejich reálných atributů. Synchronizované v kompartmentovém systému jsou právě okamžiky těchto změn. V profesi simulace systémů se říká, že změna stavu je podřízena **automatové plánovací soustavě** nebo stručněji, že v systému je automatová plánovací soustava. Přívlastek automatová navazuje obsahově na objekty matematických teorií automatů, jak je studovala matematická logika a dnes tak zvaná teoretická kybernetika nebo teoretická informatika.

Buněčné systémy se z hlediska profese počítačové simulace a vůbec teorie systémů liší od systémů kompartmentových jednak tím, že počet jejich prvků se v čase mění, a jednak tím, že v nich není automatová plánovací soustava.

Ze středoškolské biologie víme, že se buňky dělí a že to je na nich právě důležité. A víme, že buňky také mohou hynout, být likvidovány následkem nekrotických procesů. Když se buňka rozdělí, je najednou v systému o jednu buňku (tedy o jeden prvek) více, a když zhyne (odborně se říká **opustí systém**), je v buněčném systému o jeden prvek méně. Rozdělení buňky, stejně jako její zhynutí, je relativně krátký proces, z jedné buňky najednou, během nekonečně krátkého časového okamžiku, vzniknou buňky dvě. A podobně je tomu se zhynutím buňky: buňka existuje a najednou - opět v nekonečně krátkém časovém okamžiku - přestane v systému existovat. Buňky bývají v několika zvlášť důležitých biologických stavech; přechod z jednoho stavu do druhého trvá sice nějakou dobu, avšak ta je tak krátká, že se vyplatí chápat takový přechod jako okamžitou změnu. V takovémto zjednodušeném jakopojetí je život buňky průchodem několika biologickými stavy: v každém z nich buňka setrvá nějakou dobu a na jejím konci přejde do dalšího biologického stavu. Na konci jistého biologického stavu se buňka rozdělí; ona de facto zmizí a místo ní vstoupí do systému dvě nové buňky a ty žijí svými vlastními životy, tj. procházejí sice stejnými biologickými stavy, ale doby, během nichž v těchto stavech setrvávají, se obecně liší, takže obě buňky začnou brzy měnit svůj stav, každá jindy, a nakonec se rozdělí každá v jiném okamžiku.

Je tedy evidentní, že buněčný systém nemá automatovou plánovací soustavu. „Životy“ buněk (to jest přechody a setrvávání v jednotlivých stavech) nejsou synchronizovány.

Formální popis buňky by mohl vypadat asi takto. Když buňka vznikne, je v jistém biologickém stavu S_1, po jisté době t_1 přejde do stavu S_2, v něm zůstává jistou dobu t_2 a pak přejde do stavu S_3 atd., až se dostane do posledního stavu S_m, v něm setrvá po dobu t_m, pak se rozdělí a zmizí. Popis, který je zcela ekvivalentní, ale který není pro biology snadno akceptovatelný, je podobný, s výjimkou závěrečné fáze: buňka se nerozdělí, nýbrž „zplodí“ další buňku a sama vstoupí do stavu S_1 a svůj životní cyklus zopakuje (obecně s jinými hodnotami $t_1, t_2, \ldots$). Když se buňka dostane do daného biologického stavu S_i, je následující stav určen - aspoň v obvyklých buněčných systémech - jednoznačně, avšak doba t_i, po kterou buňka v tomto stavu zůstává, přesně určena není: závisí na mnoha faktorech a obvykle se chápe jako náhodná (na počítači je její velikost interpretována jako náhodné číslo).

6.2.4.2 Jazyky pro simulaci buněčných systémů

Popis života buňky je přímo ideální ukázkou toho, jak „komunikace s počítačem“ vede k vytvoření simulačního modelu buněčného systému. Dejme tomu, že v simulovaném systému chápeme buňku jako objekt, který prochází postupně stavy, jež budeme značit $G0, G1, S, G2$ a M, a v posledním z nich se rozdělí. Přitom předpokládáme, že ve stavu S zůstane buňka po pevně určenou dobu dvaceti časových jednotek, zatímco ostatní doby budou nedeterministické, splňující jisté stochastické zákony rozdělení (např. doba setrvání ve stavu $G0$ bude mít lichoběžníkové rozdělení, doba setrvání ve stavu $G1$ bude mít trojúhelníkové rozdělení a doba setrvání ve stavu $G2$ stejně jako doba setrvání ve stavu M budou mít normální rozdělení.

Taková „životní pravidla“ pro buňku bychom mohli popsat např. takto:

```
process bunka;
begin text state; loop:
      state:="G0"; hold(trapez(...));
      state:="G1"; hold(triang(...));
      state:="S"; hold(20);
      state:="G2"; hold(normal(...));
      state:="M"; hold(normal(...));
      new bunka; go to loop
end;
```

Tento popis „sděluje“ počítači následující informace. Každá buňka má jeden atribut nazvaný `state`. Může nabývat textových hodnot. Buňka začíná svůj

život tím, že si za hodnotu atributu `state` dosadí `"G0"` (můžeme tedy říci, že je v biologickém stavu $G0$), pak čeká, až se simulovaný čas zvětší o `trapez(...)`, kde `trapez` je generátor pseudonáhodných čísel s lichoběžníkovým rozdělením (v závorce jsou parametry tohoto generátoru), pak si dosadí za `state` hodnotu `"G1"` a čeká, až se hodnota simulovaného času zvětší o `triang(...)`, kde `triang` je generátor pseudonáhodných čísel s trojúhelníkovým rozdělením, a tak dále. Když si buňka dosadí za state hodnotu `"M"` a čeká uvedenou dobu, vytvoří novou buňku a svůj další život si organizuje tak, že v jeho algoritmickém popisu skočí na příkaz, který následuje za návěštím `loop`.

Takový popis něco řekne o všech buňkách, popis modelu je však nutno ještě doplnit o informaci, kolik buněk je v modelu na počátku existence modelovaného systému, co nás na modelu zajímá (tedy co má být ve výstupním souboru) a kdy má simulační pokus skončit. To vše lze udělat např. příkazy, jež se uvedou za výše uvedeným popisem životních pravidel buňky:

```
for i:=1 step 1 until 400 do new bunka;
hold(1500); outtext(...);
```

V prvním řádku se určuje, že se vytvoří nová buňka pro $i = 1, 2, \ldots, 400$; simulovaný systém bude tedy obsahovat na počátku 400 buněk. Zatímco tyto buňky budou podle svých pravidel „žít“ (tj. ovládat výpočet) a dělit se, bude řízení simulačního pokusu čekat, až simulovaný čas dosáhne hodnoty 1500. V tom okamžiku se na výstupu objeví to, co je uvedeno v příkazu `outtext`, a pak simulační pokus skončí.

6.2.4.3 Inspirace a podněty

Životní pravidla jsou velmi podobná algoritmům. Liší se od nich jen plánovacími příkazy (viz odstavec 4.2), v našem příkladě tím, co jsme vyjádřili pomocí slova `hold`. Když život konkrétní buňky narazí na takové pravidlo, přeruší se jeho interpretace v počítači a řízení výpočtu se předá obecně jiné buňce. Lze říci, že ostatní životní pravidla (tedy ta, která nepatří mezi plánovací příkazy) se provádějí jen během toho, co jsme výše nazvali změnami stavu.

Každý si jistě umí představit, jak počítač generuje pseudonáhodná čísla a přepíná automaticky mezi těmi čtyřmi sty a více buňkami, aby se ve správném pořadí realizovaly jejich stavové změny, a každý jistě ocení, jaká je to pomoc uživateli, když počítač sám všechny takové změny ve správném pořadí synchronizuje. O synchronizaci se mluví také jako o **plánování událostí** (viz odstavec 4.3.4). V modelech, kde se vyskytují příkazy podobné jako výše uvedené `hold(...)`, mluvíme o **imperativní plánovací soustavě**. Existuje ještě tak zvaná **interrogativní plánovací soustava**, která nabízí uživateli příkazy tvaru `wait until b`, kde `b` je nějaká podmínka. Prvek, který na takový příkaz narazí, čeká, dokud není tato podmínka splněna. A existuje i kombinovaná

plánovací soustava **imperativně-interrogativní**, která nabízí oba druhy příkazů. Musíme poznamenat, že v dnešní době převládá imperativní plánovací soustava, neboť ostatní dvě vyžadují příliš mnoho strojového času pro stálé testování, zda je či není daná podmínka právě splněna. V odstavci 7.2.4 se dozvíme, že plánovacím soustavám odpovídají i „stejnojmenné“ plánovací příkazy. Právě zmíněný příkaz `hold(...)` je tedy imperativní plánovací příkaz.

Vraťme se ještě k oné podobnosti mezi životními pravidly a algoritmy. Ve výše uvedeném příkladě se vyskytují dosazovací příkazy a příkaz skoku. Čtenáře může napadnout, že by někdy v životních pravidlech mohly být užitečné i další řídící struktury běžné v tradičních procedurálně orientovaných programovacích jazycích. V tomto případě je třeba říci, že záleží na konkrétním simulačním jazyku: některý může být velmi vyvinutý a připouští mnoho, jiný naopak může být jednoduchý či spíše značně specializovaný (omezený na jistou specifickou třídu systémů) a jeho algoritmické prostředky jsou mizivé.

S použitím dalších algoritmických prostředků bychom mohli rozšířit model tak, aby zahrnoval tři „větve“ životních pravidel: kromě výše popsané ještě větev týkající se buněk, které neprojdou stavem $G0$, a větev buněk, které se v závěru stavu M nedělí, nýbrž hynou.

```
process bunka;
begin text state; loop:
      if draw(0.2) then go to L;
      state:="G0"; hold(trapez(...));
    L:state:="G1"; hold(triang(...));
      state:="S"; hold(20);
      state:="G2"; hold(normal(...));
      state:="M"; hold(normal(...));
      if draw(0.4) then go to exit;
      new bunka; go to loop; exit:
end;
```

`if draw(p) then` znamená, že příkaz uvedený za `then` se neprovede vždy, nýbrž jen s pravděpodobností `p`. Uvedený příklad je velmi blízko několika jazykům pro diskrétní simulaci, konkrétně jazykům SOL, SLANG a SIMULA, pokud se používá jeho třída SIMULATION obsahující prostředky pro simulaci. Prostředky uvedených jazyků se liší navzájem a také od našeho příkladu zejména v některých symbolech a slovech, ale „vidění světa“, totiž to, že prvky systému mají svá životní pravidla, která lze chápat jako algoritmy obsahující navíc plánovací příkazy, je v případě uvedených jazyků stejné.

6.2.4.4 Jazyk CELLSIM

Nejčastější důvod pro simulaci buněčného systému je snaha dozvědět se, jak se bude vyvíjet počet všech buněk, resp. počet buněk, které budou v určitém biologickém stavu, nebo jakého množství dosáhnou v konkrétně dané době.

Můžeme vznést následující námitku. Jak tento počet zjistíme, to budeme nějak sledovat všechny prvky, které se v modelu vyskytnou, nebo snad všechny struktury právě reprezentované v počítači a zjišťovat, zda jsou to buňky a v jakém jsou stavu? To je přece zdlouhavé a v podstatě hloupé!

Odstranit tuto potíž není obtížné, používáme-li jazyka podobného těm zmíněným v odstavci 6.1.2. Tyto (a jiné) jazyky pro diskrétní simulaci totiž nabízejí prostředky pro práci se seznamy či s množinami (tj. se seznamy, u nichž se ignoruje uspořádání), a tak v případě simulace buněčných systémů prostě nahradíme textovou interpretaci biologického stavu příslušností do množiny buněk, které v takovém biologickém stavu jsou. Zavedeme tedy pět množin $G0, G1, G2, S$ a M a místo toho, abychom buňce přiřazovali stav-text jako její atribut, budeme ji vkládat do příslušné množiny. Text popisující celý systém bude nyní mít tento tvar:

```
set G0,G1,G2,S,M;
process bunka;
begin loop:
      if draw(0.2) then go to L;
      into(G0); hold(trapez(...));
    L:into(G1); hold(triang(...));
      into(S); hold(20);
      into(G2); hold(normal(...));
      into(M); hold(normal(...));
      if draw(0.4) then go to exit;
      new bunka; go to loop; exit:
end;
for i:=1 step 1 until 400 do new bunka;
hold(1500);
for R:=G0,G1,S,G2,M do outint(cardinal(R));
```

V prvním řádku se zavádí pod uvedenými jmény pět množin. `into(...)` znamená, že daný prvek vstoupí do množiny uvedené v závorkách. Téměř všechny jazyky pro diskrétní simulaci připouštějí, aby byl prvek vždy nejvýše v jedné množině, takže pokud prvek v nějaké množině je a má vstoupit do jiné, tak tu první automaticky opustí. Nemusíme se tedy obávat, že se buňka bude po jisté době vykytovat ve všech pěti množinách.

Poslední řádek sděluje, že se provede cyklus postupně přes všech pět množin

a pro každou z nich se nechá vystoupit počet jejích prvků; `cardinal(...)` je kardinalita čili mohutnost množiny uvedené v závorkách.

Dále uvidíme, že simulační jazyky uvažovaného typu se hodí např. i pro simulaci výrobních systémů, tj. systémů, v nichž nejsou buňky, nýbrž stroje, roboty, sklady, obrobky, nástrojové palety apod. Jde tedy o jazyky s širokým spektrem použití. Bylo vytvořeno několik simulačních jazyků vhodných právě pro odborníky v oblasti histologie. Tyto jazyky nevyžadují algoritmické cítění a ovšem nejsou vhodné ani pro simulaci výrobních systémů.

Uveďme zde jeden z těchto jazyků nazvaný CELLSIM (zkratka slov *cell simulation*) [22, 12]. Ten vyžaduje na uživateli, aby buněčný systém popsal ve třech odstavcích.

V jednom, který je uvozen slovem `FLOW` (anglicky tok), se definuje, jak buňka přechází z jednoho biologického stavu do druhého: odstavec se skládá z vět tvaru `A - B(...)`, které říkají, že ze stavu `A` přechází buňka do stavu `B`, a to s pravděpodobností uvedenou v závorkách. (Jde-li o jistotu, čili o pravděpodobnost rovnou jedné, lze do závorky napsat slovo `ALL`.)

V druhém odstavci se formulují pravidla pro to, jak dlouho buňka v jednotlivých stavech setrvává. Odstavec je uvozen textem `TIME IN STATES`, za nimž následují věty tvaru `A:E`, kde `A` je název stavu a `E` je výraz určující dobu setrvání.

Na konci každého odstavce musí být středník a věty se oddělují čárkami. Na pořadí vět nezáleží.

V třetím odstavci se definují počáteční podmínky a stav, v němž se buňka dělí. Věta tvaru `INITIAL:p(A)` říká, že na počátku simulačního pokusu bude v systému `p` buněk, a to ve stavu `A`. Věta tvaru `PROLIFERATION:A(n)`, říká že se buňka dělí ve stavu `A`, a to na `n` buněk. (Jazyk CELLSIM dovoluje simulovat i buňky, které se dělí na více než dvě.) Věta tvaru `END:t` říká, že se má simulační pokus ukončit, když simulovaný čas dosáhne hodnoty `t`, a pak se postupně pro každý stav nechá vystoupit počet buněk, které v okamžiku ukončení simulačního pokusu v tomto stavu byly.

Příklad uvedený výše bychom tedy mohli v jazyku CELLSIM zapsat takto:

```
FLOW G0 - G1(ALL), G1 - S(ALL), G2 - M(ALL), S - G2(ALL),
     M - G0(0.4), M - G1(0.2), M - T(0.4);
TIME IN STATES G0:TRAPEZOID(...), G1:TRIANGLE(...),
     G2:NORMAL(...), S:20.0, M:NORMAL(...);
INITIAL:80(G1), INITIAL:320(G0), PROLIFERATION:M(2),
     END:1500;
```

Jazyk CELLSIM povoluje modelovat v jediném systému více druhů buněk. Totéž dovolují i simulační jazyky, jejichž vlastnosti jsou uvedeny v odstavci 6.2.3. Ty povolují modelovat i vzájemné ovlivňování buněk - např. doba, po kterou

je buňka v jistém stavu, může záviset na počtu jiných buněk v nějakém jiném stavu nebo na koncentraci nějaké látky.

6.3 Celulární automaty

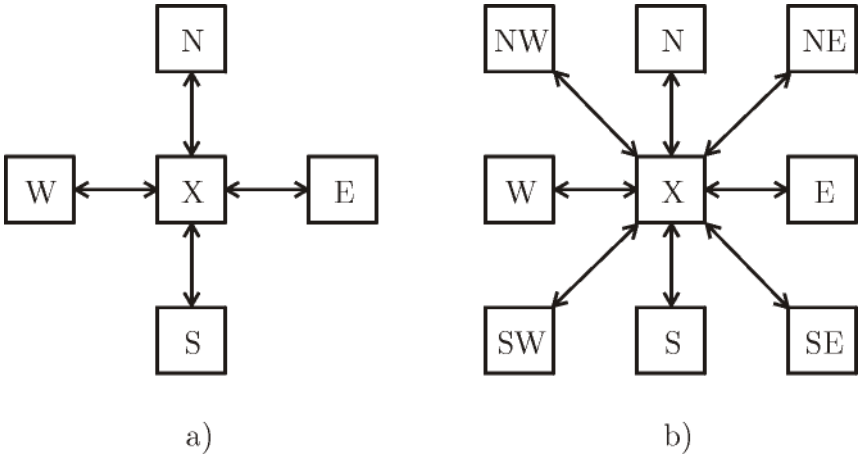
6.3.1 Základní pojmy

Pod pojmem **celulární automat** (CA) se obvykle rozumí nekonečně mnoho exemplářů nějakého konečného automatu propojených určitým uniformním způsobem [19]. Jednotlivé automaty s konečným počtem stavů se nazývají buňkami. Každá buňka CA je propojena s několika buňkami sousedními, které tvoří **okolí** dané buňky. Všechny buňky pracují synchronně, což znamená, že změny stavu, k nimž dochází v diskrétních časových krocích (taktech), nastávají vždy ve všech buňkách současně. Přitom stav kterékoli buňky v následujícím taktu je určen současným stavem této buňky a buněk, které jsou v jejím okolí. Předpis, jímž se definuje stav uvažované buňky v následujícím taktu, se nazývá **přechodová funkce** této buňky.

Uveďme alespoň dva jednoduché příklady propojení buněk v CA (viz obr. 6.2).

Buňky na obr. 6.2a tvoří čtvercovou mřížku (síť), přičemž každá buňka (např. X) je propojena se čtyřmi nejbližšími sousedy ve směrech severním (N), východním (E), jižním (S) a západním (W). Buňky označené N, E, S, a W tvoří tzv. **Von Neumannovo okolí** buňky X. Stav buňky X v následujícím taktu je obecně určen přítomným stavem celé pětice uvažovaných buněk.

Obr. 6.2b představuje rovněž čtvercovou mřížku buněk, v níž je každá buňka napojena na osm buněk sousedních. Kromě čtyř nejbližších sousedů (N, E, S, W) zahrnuje okolí dané buňky X také čtyři druhé nejbližší sousedy ve směrech severovýchodním (NE), jihovýchodním (SE), jihozápadním (SW) a severozápadním (NW). Okolí tohoto typu se nazývá **okolí Moorovo**.



Obrázek 6.2: Příklady propojení buněk v CA

V aplikacích se zpravidla pracuje s dvourozměrnou čtvercovou mřížkou buněk, i když se nevylučuje možnost využití mřížky jiné symetrie (např. hexagonální).

Každá buňka CA je v daném okamžiku právě v jednom z konečné množiny stavů. Buňky CA mají obecně jeden význačný stav, tzv. **klidový stav**. Jsou-li daná buňka i celé její okolí v klidovém stavu, pak i v následujícím taktu zůstane tato buňka ve stavu klidovém. Proto také CA, jehož všechny buňky se nacházejí v klidovém stavu, zůstává trvale beze změny. Pokud se má stav CA měnit, musí tento automat obsahovat tzv. **aktivní buňky**, tj. buňky v jiném než klidovém stavu.

Konfigurace CA je určena nejen počtem a rozložením aktivních buněk, ale i tím, v jakém stavu se každá z nich nachází. V teorii i aplikacích se sledují především konečné konfigurace. Přitom je zřejmé, že od dané konečné konfigurace může CA přejít vždy jen k nějaké konečné konfiguraci.

6.3.2 Modelování a simulace pomocí CA

V uvažovaném případě jde o nahrazení reálného systému jeho CA modelem a takové experimentování s tímto modelem, které směřuje k získání informací o původním zkoumaném systému. Simulační CA model je diskrétní dynamický systém vymezený takto (viz [14, 15]):

- stavový prostor je pravidelná jedno- nebo dvourozměrná mřížka tvořená jednotně propojenými buňkami, přičemž každá buňka se v daném okamžiku nachází právě v jednom z konečné množiny stavů;
- stavy všech buněk se mění v diskrétních okamžicích (taktech), a to vždy synchronně;
- změny stavů buněk jsou definovány pomocí množiny přechodových funkcí, přičemž stav kterékoli buňky v následujícím taktu závisí na současném stavu této buňky a jejího okolí.

Implementace CA modelu na počítači (vytvoření počítačového modelu) pak zahrnuje následující kroky:

1. vytvoření matice, jež reprezentuje počáteční konfiguraci CA modelu,
2. definování množiny přechodových funkcí, které realizují změny konfigurace CA modelu (prvků maticové reprezentace) v jednom taktu,
3. postupná aplikace těchto přechodových funkcí na posloupnost konfigurací CA modelu (posloupnost maticových reprezentací).

Každá buňka CA má v daném okamžiku specifikovanou hodnotu (celé číslo, reálné číslo, symbol nebo seznam prvků libovolného typu), jež jednoznačně určuje stav této buňky. Výběr typu hodnoty se přitom podřizuje povaze simulovaného systému.

Při aplikacích CA modelů hraje významnou roli otázka adekvátní volby okrajových podmínek. V případě CA modelů nekonečných systémů (systémů bez hranic) se nejčastěji používají **periodické okrajové podmínky**, což znamená, že se konfigurace CA mřížky periodicky opakuje ve všech dimenzích.

Při simulaci systémů, v nichž se objekty pohybují v omezeném prostoru, se často volí jiný přístup. Hraniční buňky mají speciální hodnotu, odlišnou od hodnot buněk vnitřních, a tato hodnota se v průběhu simulace nemění. Přechodové funkce pak určují, co se děje s hodnotami buněk v okolí takových hraničních buněk (absorbující nebo reflektující hranice).

V některých případech se osvědčují i pohyblivé hranice. Velikost CA mřížky se přitom v každém taktu mění tak, že vše podstatné se odehrává uvnitř mřížky a vliv hraničních buněk je nepodstatný.

Přechodová funkce libovolné buňky X je v případě Von Neumannova okolí obecně funkcí pěti argumentů, z nichž jeden reprezentuje stav této buňky a zbývající stavy buněk z jejího okolí. Přechodové funkce mohou mít i stochastický charakter.

Formulace přechodových funkcí představuje zřejmě nejdůležitější krok při konstrukci simulačního modelu. Přitom ovšem neexistují žádné obecně platné zásady pro jejich výběr.

Přechodové funkce se aplikují opakovaně na všechny buňky CA mřížky a tím se simuluje vývoj zkoumaného systému. Simulace zpravidla končí po realizaci předem zadaného počtu taktů. Ukončení simulace je také možno vázat na splnění jisté podmínky, např. dosažení stacionárního stavu, kdy další použití přechodových funkcí již nevede k žádným změnám v konfiguraci CA.

Téměř ideálním prostředkem pro zápis simulačních CA modelů je jazyk *Mathematica*. Podrobnosti o programování v jazyku *Mathematica* nalezne čtenář v učebnici [32].

6.3.3 Ilustrativní příklad

Uvažujme CA model pro šíření epidemie nějaké nakažlivé choroby v relativně početné populaci tvořené n^2 jedinci [15]. Předpokládejme, že se nákaza šíří kontaktem mezi infikovanými a vnímavými jedinci. Nechť průměrná doba trvání nemoci je a dnů a každý jedinec je po překonání nemoci ještě b dnů imunní. Dále nechť m udává tu část populace, která se nachází na počátku (v čase 0) ve stavu infekčnosti nebo imunity. Infekční a imunní jedinci jsou přitom umístěni v populaci zcela náhodně.

Stav každého jedince v populaci bude popsán přirozeným číslem takto:

0	vnímavý jedinec,
$1, 2, \ldots, a$	infikovaný jedinec,
$a+1, a+2, \ldots, a+b$	imunní jedinec.

K reprezentaci počáteční konfigurace CA mřížky použijeme matice typu $n \times n$, v níž budou náhodně rozloženy prvky indikující nakažené a imunní jedince.

Stav celé populace se bude každým dnem měnit podle následujících pravidel:

- Vnímavý jedinec se stane infekčním, jestliže alespoň jeden z jeho sousedů je infekční.
- Vnímavý jedinec, jenž nemá ve svém okolí žádné infekční jedince, zůstává vnímavým.
- Jedinec, který byl imunní po dobu b dnů, se stává opět vnímavým vůči uvažované nákaze.
- Hodnota popisující stav jedince, který je buď infekční nebo imunní po dobu kratší než b dnů, se zvyšuje o jednotku. (Podle tohoto pravidla každý jedinec, jenž byl infekční po dobu a dnů, přechází do kategorie imunních jedinců).

Tato pravidla určují tvar přechodových funkcí. Pokud se omezíme na okolí Von Neumannova typu, budou přechodové funkce pro buňku X obecně definovány takto

```
spread[x_, n_, e_, s_, w_] := <výraz>,
```

kde argumenty v uvedeném pořadí reprezentují stav buňky X a stavy okolních buněk N, E, S, W ve schématu na obr. 6.2a.

Popsaný CA model je možno implementovat např. takto:

```
epidemic[n_, m_, a_, b_, t_] :=
Module[{initpop, spread, VonNeumann},
initpop = Table[Floor[1 + s - Random[]] *
Random[Integer, {1, a + b}], {n}, {n}];
spread[0 | (a + b), _, _, _, _] :=   0;
spread[x_?Positive, _, _, _, _] :=   x + 1;
spread[0, u_, v_, w_, z_] := 1 /;
MemberQ[Range[a], u | v | w | z];
VonNeumann[func_, lat_] :=
MapThread[func, Map[RotateRight[lat,  #]&,
{{0, 0},  {1, 0}, {0, -1}, {-1, 0}, {0, 1}}], 2];
NestList[VonNeumann[spread, #]&, initpop, t]]
```

Celý program je uživatelskou funkcí s pěti argumenty (argument `t` reprezentuje počet taktů). Vestavěná funkce `Module` umožňuje definovat počáteční konfiguraci (`initpop`), přechodové funkce (`spread`) a anonymní funkci `VonNeumann` jako lokální právě v tomto programu. Následují definice přechodových funkcí, jež reprezentují stanovená pravidla pro šíření epidemie v jednom taktu. Anonymní funkce `VonNeumann` zajišťuje synchronní aplikaci přechodových funkcí na CA mřížku. Vestavěná funkce `NestList` pak generuje posloupnost výsledků postupné aplikace anonymní funkce `VonNeumann`. Simulace končí po realizaci t taktů.

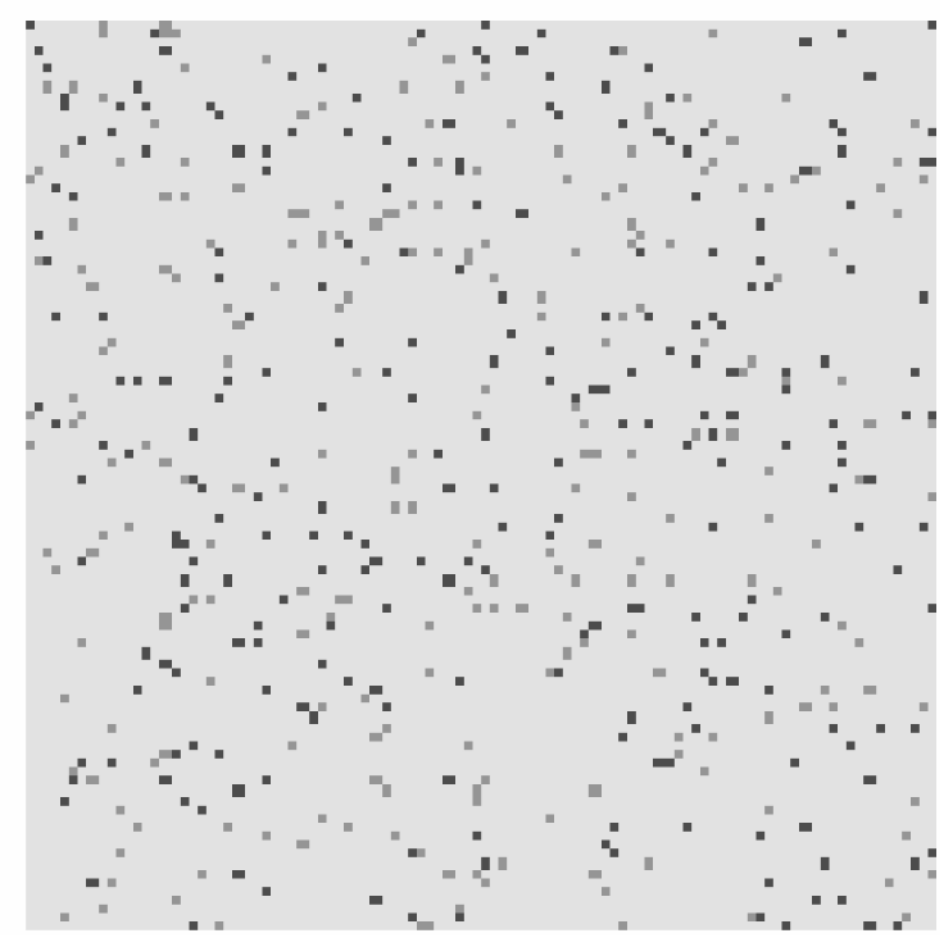
Vstupními veličinami jsou: počáteční konfigurace, velikost populace (n^2), podíl infekčních a imunních jedinců (m), průměrná délka nemoci (a), průměrná délka získané imunity (b) a počet taktů (t).

Vlastní výpočet se realizuje po zadání např.

```
SeedRandom[11];
epidemic[100, 0.05, 8, 8, 200] //MatrixForm
```

Mnohem názornější je ovšem grafický výstup, kdy jednotlivé stavy buněk jsou rozlišeny barevně. V takovém případě se použije modifikovaného zadání

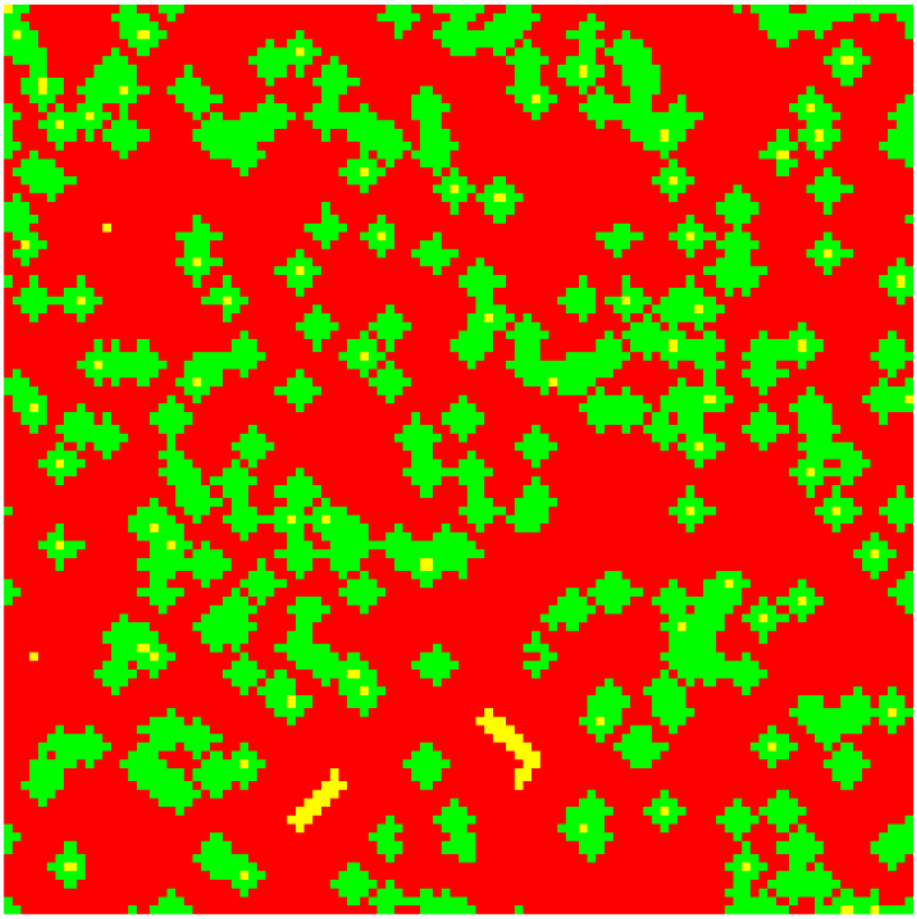
```
SeedRandom[11];
showepidemic[list_, opt__] :=
Map[Show[Graphics[RasterArray[Reverse[list[[#]] /.
0 -> RGBColor[1,1,0], 1 -> RGBColor[1,0,0],
2 -> RGBColor[1,0,0], 3 -> RGBColor[1,0,0],
4 -> RGBColor[1,0,0], 5 -> RGBColor[1,0,0],
6 -> RGBColor[1,0,0], 7 -> RGBColor[1,0,0],
8 -> RGBColor[1,0,0], 9 -> RGBColor[0,1,0],
10 -> RGBColor[0,1,0], 11 -> RGBColor[0,1,0],
```



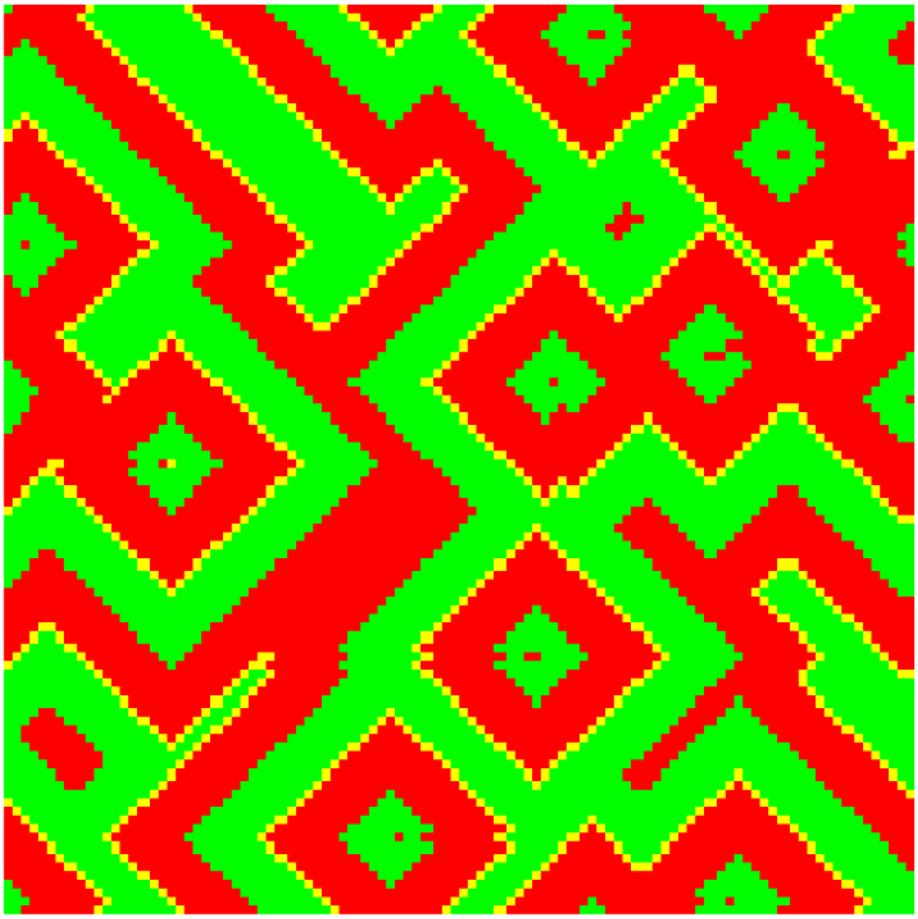
Obrázek 6.3: Počáteční konfigurace

```
12 -> RGBColor[0,1,0], 13 -> RGBColor[0,1,0],
14 -> RGBColor[0,1,0], 15 -> RGBColor[0,1,0],
16 -> RGBColor[0,1,0]}]]], opt]&,
Range[Length[list]]];
showepidemic[epidemic[100, 0.05, 8, 8, 200],
{AspectRatio -> Automatic}]
```

Obrázky 6.3 - 6.5 představují v uvedeném pořadí grafické výstupy pro počáteční konfiguraci zvoleného CA modelu a jeho konfigurace po dokončení 10 a 200 taktů.



Obrázek 6.4: Konfigurace po 10 taktech



Obrázek 6.5: Konfigurace po 200 taktech

6.3.4 Využití CA

CA jsou mimořádně vhodným prostředkem pro modelování a simulaci diskrétních dynamických systémů, zejména takových, v nichž se interakce daného prvku omezují na jeho nejbližší okolí. Proto se CA modely nejlépe osvědčují při modelování procesů v biologických a ekologických systémech (např. modely systémů dravec – kořist, epidemiologické modely, modely šíření lesních požárů). CA modely se ovšem uplatňují i při řešení celé řady jiných fyzikálních, chemických, technických a sociálních problémů (dopravní problémy, difúze, adsorpce a desorpce, chemotaxie apod.). Ilustrativní příklad z oblasti epidemiologie ukazuje, jak snadno lze implementovat CA model na personálním počítači pomocí programovacího jazyka *Mathematica*.

6.4 Lindenmayerovy systémy

Lindenmayerovy systémy nebo ve zkratce **L-systémy** představují zvláštní typ symbolického dynamického systému vhodného pro popis a geometrickou interpretaci systémové evoluce. Poprvé byly použity A. Lindenmayerem v roce 1968 pro modelování biologického růstu. Teorie L-systémů je popsána v monografii Hermana a Rozenberga [18]; pro úvodní seznámení s problematikou můžeme doporučit studijní materiál D. J. Wrighta [52], dostupný v elektronické podobě na Internetu.

6.4.1 Základní pojmy

Začneme definicí jednotlivých složek L-systémů. **Abeceda** je libovolná konečná množina $V \neq \emptyset$ formálních symbolů, zpravidla písmen, případně jiných znaků.

Znakové řetězce se nazývají **slova**. Speciálním případem slova je **prázdné slovo** značené $\emptyset$. Množinu všech přípustných slov nad abecedou V označíme V^*. Délka $\mid w \mid$ slova $w \in V^*$ udává počet znaků ve slově.

Axiom (**iniciátor**) je znakový řetězec $\omega \in V^*$.

Produkční pravidlo (také **přepisovací** nebo **odvozovací pravidlo**) je zobrazení nějakého symbolu $a \in V$ na nějaké slovo $w \in V^*$. Přepisovací pravidlo má následující syntax:

$$p : a \longrightarrow w$$

Připouštějí se i taková pravidla, která zobrazují znak a na prázdné slovo, resp. na znak a. Jestliže znak $a \in V$ nemá žádné explicitně uvedené produkční pravidlo, předpokládáme, že se zobrazuje sám na sebe. V takovém případě reprezentuje znak $a \in V$ **konstantu** příslušného L-systému.

Jednoduchým příkladem L-systému je **Fibonacciho L-systém**; jeho složky jsou definovány takto:

$$V = \{a, b\}$$

$$\omega = a$$

$$p_1 : a \longrightarrow b$$

$$p_2 : b \longrightarrow ba$$

Evoluce takového L-systému je definována jako posloupnost generací $\{g_n\}$, $n = 0, 1, \ldots,$ v níž každá generace $g_n \in V$ vznikne z bezprostředně předcházející generace g_{n-1} aplikací všech produkčních pravidel na každý ze znaků generace g_{n-1}.

Výchozí generace g_0 je zřejmě reprezentována axiomem ω. Několik prvních generací Fibonacciho L-systému je uvedeno v tabulce 6.1.

Tabulka 6.1: Generace Fibonacciho L-systému

g_0	a
g_1	b
g_2	ba
g_3	bab
g_4	$babba$
g_5	$babbabab$
g_6	$babbababbabba$
g_7	$babbababbabbababbabab$

Můžeme si představit, že symboly a, b označují dva typy jedinců nějaké populace: a dítě, b dospělého jedince. Uvedená produkční pravidla je pak možno interpretovat takto: Po jedné generaci jedinec typu a (dítě) dospěje a stane se z něj jedinec typu b. Tento dospělý jedinec je schopen produkovat v každé generaci právě jedno dítě. Popsaný symbolický dynamický systém modeluje velmi jednoduchým způsobem růst populace. Pro počet jedinců v n-té generaci zřejmě platí $\mid g_n \mid = F_n$, kde F_n je Fibonacciho číslo, přesněji člen Fibonacciho posloupnosti definované rekurentním předpisem

$$F_0 = F_1 = 1,$$

$$F_{n+2} = F_{n+1} + F_n \quad \text{pro} \quad n \geq 0.$$

6.4.2 Typy L-systémů

Fibonacciho systém je příkladem tzv. **bezkontextového** (**context-free**) **L-systému**, protože jeho produkční pravidla se aplikují na jednotlivé symboly bez ohledu na to, v jakém kontextu se vyskytují. Pokud produkční pravidla závisejí také na tom, jaké má uvažovaný symbol sousedy (okolí), pak jde o **kontextově závislé** (**context-sensitive**) **L-systémy**. Uvedeme jednoduchý příklad takového systému.

$$V = \{a, b, c\}$$

$$\omega = bb$$

$$p_1 : a(> \emptyset) \longrightarrow b$$

$$p_2 : a(> b) \longrightarrow \emptyset$$

$$p_3 : b(> a) \longrightarrow c$$

$$p_4 : b(> b) \longrightarrow ba$$

Tabulka 6.2: Generace kontextově závislého L-systému

g_0	bb
g_1	$babab$
g_2	ccb
g_3	b

$$p_5 : c \longrightarrow \emptyset$$

Tento L-systém má již složitější dynamiku. Předpokládejme, že význam symbolů je následující: a značí dítě, b dospělého (produktivního) jedince a c jedince starého. Pokud jedinec typu a nemá zprava žádného souseda (takový je přesný význam notace $> \emptyset$), normálně po jedné generaci dospěje a přemění se v jedince typu b. Jestliže však jedinec typu a sousedí zprava s jedincem typu b, pak po jedné generaci zanikne. Vyskytuje-li se jedinec b v kontextu s jedincem a zprava, potom po jedné generaci zestárne a přemění se v jedince typu c. Pokud se vedle sebe nacházejí dva jedinci typu b, dochází k reprodukci a vzniká „mezi nimi“ jedinec typu a. Jedinec typu c po jedné generaci zanikne. Evoluce takového L-systému je znázorněna v tabulce 6.2.

Z tabulky 6.2 je zřejmé, že vývoj popsaného L-systému končí u třetí generace, všechny další generace jsou stejné. V uvedených dvou příkladech existovalo pro každý symbol (s ohledem na jeho kontext) právě jediné produkční pravidlo. Takové L-systémy se nazývají **deterministické**; jejich evoluce (posloupnost generací) je definována jednoznačně. Jestliže pro daný symbol (např. a) existuje více produkčních pravidel (např. $a \longrightarrow b_1$ a $a \longrightarrow b_2$), je třeba zadat, s jakou pravděpodobností se tato konkurenční pravidla uplatní. Systémy tohoto typu se nazývají **stochastické**.

6.4.3 Grafické aplikace L-systémů

Jestliže formálním symbolům v definici L-systému přiřadíme vhodnou geometrickou interpretaci, mohou výsledné L-systémy produkovat zajímavé geometrické objekty - fraktály. Uvedeme aspoň jeden příklad z oblasti tzv. želví grafiky (turtle graphics). Podrobnosti nalezne čtenář v monografii S. Paperta [40]. Základním objektem na obrazovce je želva, jež má dva atributy: souřadnice polohy a orientaci (směr pohybu). Při svém pohybu zanechává želva na obrazovce stopu. Nechť d je vzdálenost, kterou „urazí“ v jednom kroku (tzv. délka kroku) a δ úhel natočení, přičemž

$$\delta = \frac{360^0}{n}, \quad \text{kde} \quad n \quad \text{je nějaké přirozené číslo.}$$

Zavedeme následující definice:
F: pohyb vpřed o jeden krok při aktuální orientaci,
$+$: natočení o úhel δ v kladném směru (proti pohybu hodinových ručiček),
$-$: natočení o úhel δ ve směru pohybu hodinových ručiček.

Zvolíme $\delta = 60^0$ a definujeme L-systém:

$$V = \{F, +, -\}$$

$$\omega = F$$

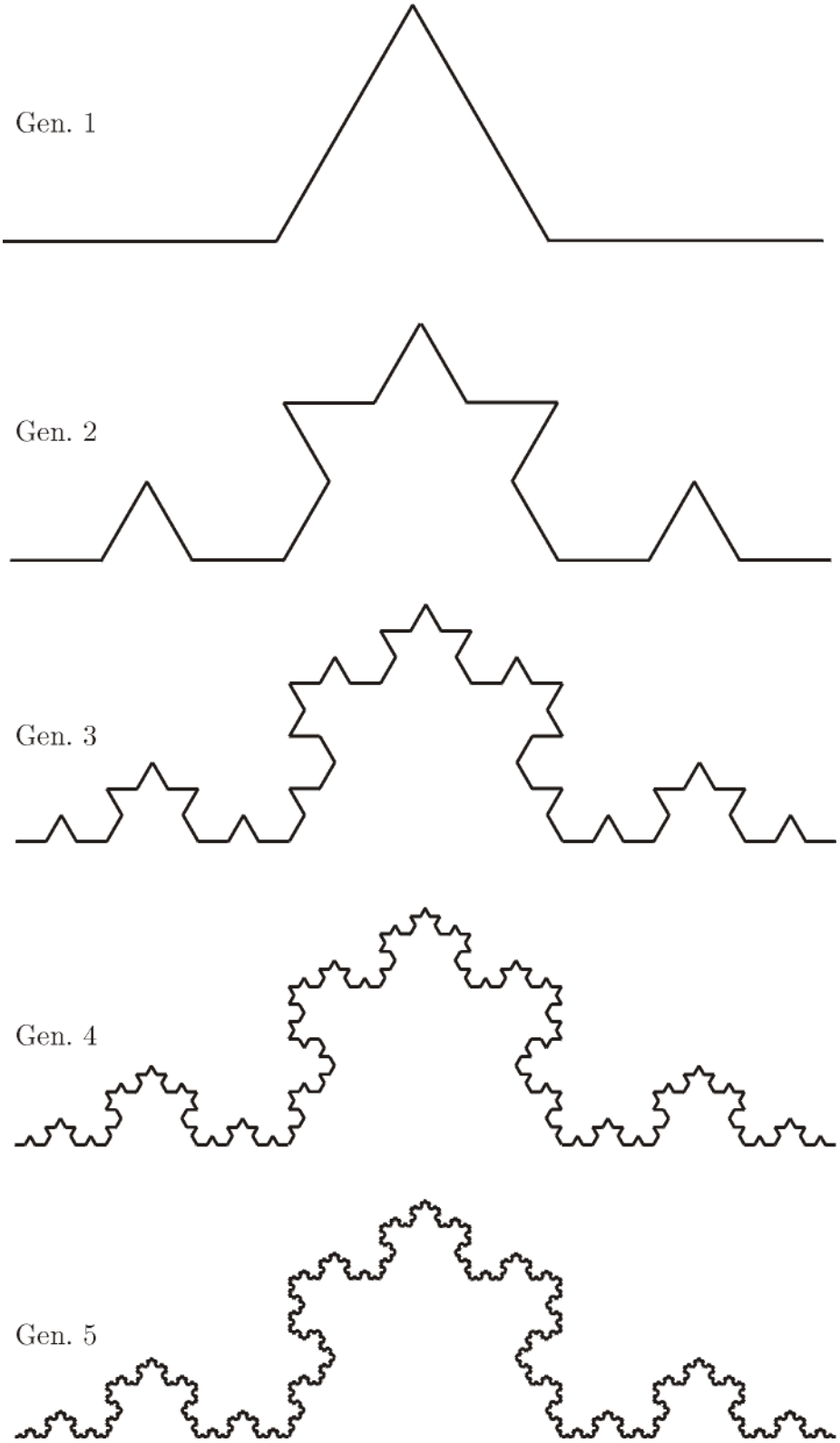
$$p_1 : F \longrightarrow F + F - -F + F$$

Předpokládejme, že na počátku je želva orientována horizontálně zleva doprava. Generace 0 je zřejmě reprezentována úsečkou délky δ. Dále zvolíme škálový faktor $= \frac{1}{3}$ tak, aby počáteční a koncové body postupně generovaných křivek zůstávaly na stejném místě. Pak lze produkční pravidlo uvažovaného L-systému interpretovat takto: vyjmout prostřední segment úsečky (o délce $\frac{d}{3}$) a nahradit jej horní částí rovnostranného trojúhelníku sestrojeného právě nad vyjmutým segmentem.

Prvních pět generací uvažovaného L-systému je schematicky znázorněno na obr. 6.6.

Tento L-systém produkuje fraktál známý pod názvem **křivka Kochové**. Počet segmentů tohoto fraktálu narůstá velmi rychle, n-tá generace je tvořena 4^n segmenty. V limitním případě ($n \to \infty$) dostaneme fraktální křivku, která nemá definovanou tečnu v žádném svém bodě.

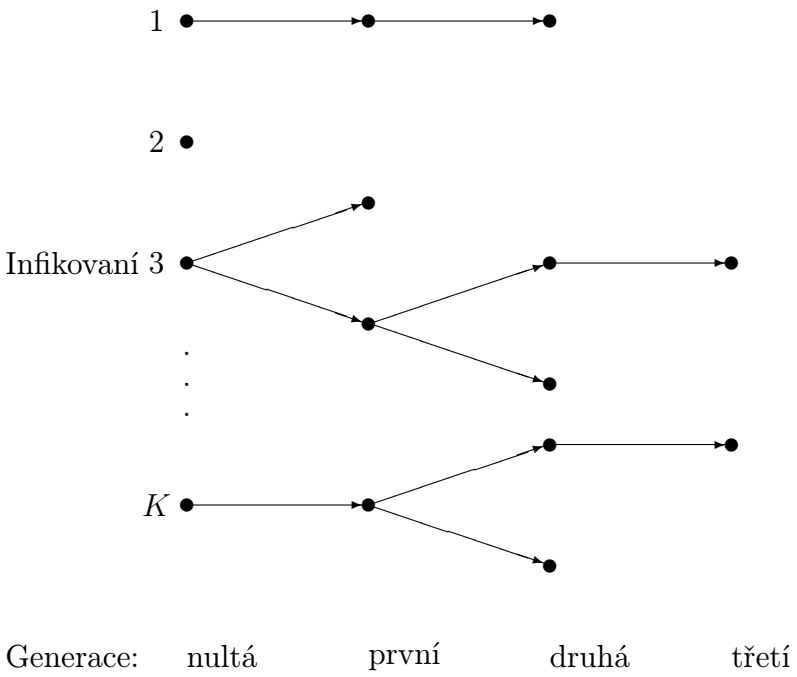
Podobným způsobem je možno generovat řadu dalších fraktálů, např. dračí křivku (Dragon curve) nebo Sierpiňského trojúhelník.



Obrázek 6.6: Generování křivky Kochové

6.5 Epidemiologické modely

V této kapitole je vysvětlena aplikace modelu větvícího se procesu v epidemiologii [13]. Z pohledu takového modelu může být malá epidemie zobrazena pomocí orientovaného grafu, jehož uzly reprezentují infikované jedince a hrany odpovídají cestám přenosu infekce (viz obr. 6.7). Epidemie skončí, jestliže všechny větve tohoto orientovaného grafu sestávají z konečného počtu hran.



Obrázek 6.7: Schéma modelu malé epidemie

Uvažovaný model vychází z těchto předpokladů:

- na počátku existuje právě K infikovaných jedinců, kteří importují nemoc do zdravé populace;
- doba infekčnosti je spojitou náhodnou veličinou, jež má exponenciální rozdělení s parametrem μ;
- ke kontaktům mezi nakaženými a vnímavými jedinci dochází zcela náhodně, přičemž průměrný počet nových případů přímo infikovaných jednou nakaženou osobou za jednotku času je roven λ;

- infikovaní jedinci jsou navzájem nezávislí, což znamená, že počet případů generovaných jednou nakaženou osobou je nezávislý na počtu případů generovaných kteroukoli jinou nakaženou osobou.

6.5.1 Základy teorie větvících se procesů

V této části připomeneme principy teorie diskrétních větvících se procesů (viz např. [4]). Uvažujme populaci, jež vytváří diskrétní generace v konstantních časových intervalech. Nechť X_i představuje náhodnou veličinu udávající počet jedinců v i-té generaci, přičemž na počátku (v nulté generaci) existuje právě jediné individuum, tj. $X_0 = 1$. Nechť tento jedinec generuje j nových jedinců následující generace s pravděpodobností p_j, $j = 0, 1, 2, \ldots$. Pak vytvořující funkce celočíselné náhodné veličiny X_1, udávající velikost populace v první generaci, má tvar

$$P(s) = \sum_{j=0}^{\infty} p_j s^j.$$

Každé individuum první generace vytvoří j potomků se stejným rozdělením pravděpodobností $\{p_j\}$, a to nezávisle na kterémkoli jiném individuu. Proto je náhodná veličina X_2, reprezentující velikost populace ve druhé generaci, součtem X_1 sdruženě nazávislých náhodných veličin s identickou vytvořující funkcí $P(s)$, takže má složené rozdělení s vytvořující funkcí

$$P_2(s) = P(P(s)).$$

Obecně platí

$$P_n(s) = P_{n-1}(P(s)) = P(P_{n-1}(s)) \quad \text{pro všechna } n > 1. \tag{6.1}$$

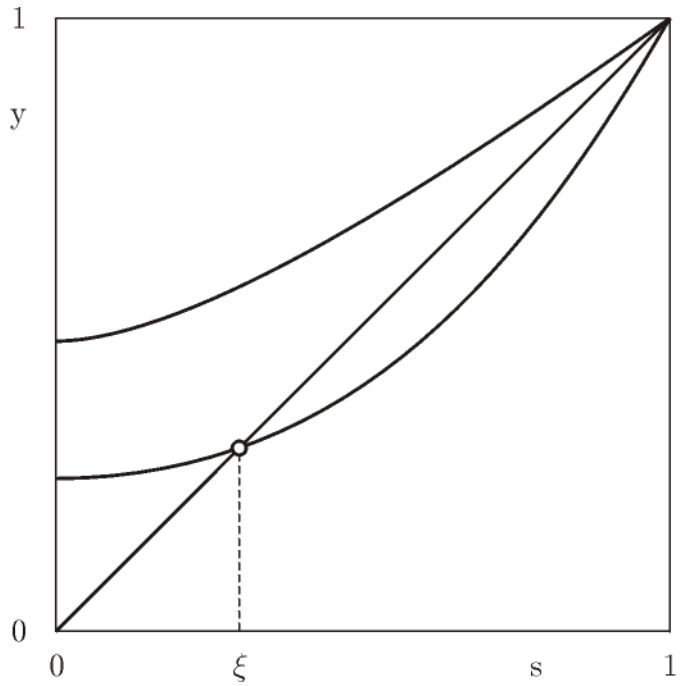
Kdyby ovšem nultá generace sestávala z K jedinců, pak by každý z těchto jedinců generoval zcela nezávisle větvící se proces právě popsaného typu a vytvořující funkce pro celočíselnou náhodnou veličinu X_n by měla tvar $[P_n(s)]^K$.

Uvažujme nyní pravděpodobnost x_n toho, že se větvící proces $\{X_n\}$, vycházející z jediného individua nulté generace, zastaví dříve, než dosáhne n-té generace. V triviálním případě $p_0 = 0$ platí zřejmě $x_n = 0$ pro všechna $n \geq 1$ a extinkce větvícího se procesu není možná. Nechť tedy $0 < p_0 \leq 1$. V takovém případě má posloupnost $\{x_n\}$ následující vlastnosti.

1. Pravděpodobnosti x_n rostou s hodnotou n, tj.

$$0 < x_1 < x_2 < \ldots < x_n < x_{n+1} < \ldots < 1.$$

2. Nultá generace obsahuje právě jedno individuum, proto $x_0 = 0$. Pravděpodobnost, že toto individuum nevytvoří žádného bezprostředního potomka je rovna p_0, takže $x_1 = p_0$.



Obrázek 6.8: Ilustrace k řešení rovnice (6.2)

3. Posloupnost $\{x_n\}$ je rostoucí a omezená, proto musí mít vlastní limitu. Jelikož zřejmě platí

$$x_n = P_n(0) = P(P_{n-1}(0)) = P(x_{n-1}),$$

musí tato limita ξ vyhovovat funkcionální rovnici

$$\xi = P(\xi). \tag{6.2}$$

Vytvořující funkce $P(s)$ i její derivace $P'(s)$ obsahují pouze kladné členy a musí tedy růst v intervalu $0 < s \leq 1$. Křivka $y = P(s)$ je konvexní a protíná přímku $y = s$ nejvýše ve dvou bodech, z nichž jedním je bod $[1, 1]$ (viz obr. 6.8).

Lze poměrně snadno dokázat (viz [13]), že nutná a postačující podmínka pro existenci kořenu $\xi < 1$ rovnice (6.2) má tvar $P'(1) = \theta > 1$, kde θ značí střední hodnotu počtu bezprostředních potomků jednoho individua. V tomto případě křivka $y = P(s)$ vychází z bodu $[0, p_0]$, kde $p_0 = Pr\{X_1 = 0\}$, protíná přímku $y = s$ v bodě $[\xi, P(\xi)]$ a leží stále pod ni, až dosáhne bodu $[1, 1]$. Je-li naopak $P'(1) \leq 1$, pak křivka $y = P(s)$ leží v celém intervalu $(0, 1)$ nad přímkou $y = s$, a proto neexistuje žádný kořen $\xi < 1$ rovnice (6.2).

Je-li tedy $\theta > 1$, pak kořen $\xi < 1$ rovnice (6.2) udává jednoznačně pravděpodobnost extinkce uvažované populace po nějakém konečném počtu generací.

Platí-li však $\theta \leq 1$, potom má rovnice (6.2) jediný kořen $\xi = 1$ a větvící se proces $\{X_n\}$ se ukončí s jistotou.

Uvedený výsledek se snadno rozšíří na případ, kdy nultou generaci netvoří jediné individuum, ale $K > 1$ vzájemně nezávislých individuí. V takovém případě je pravděpodobnost extinkce všech K generačních linií rovna jednoduše ξ^K. Výraz $1-\xi^K$ pak udává pravděpodobnost, že se aspoň jedna z generačních linií bude úspěšně rozvíjet.

6.5.2 Celková velikost epidemie

Každý infikovaný jedinec zůstává zdrojem nákazy po jistou dobu T (tzv. dobu infekčnosti), kde T představuje spojitou náhodnou veličinu mající exponenciální rozdělení s parametrem μ a hustotou

$$f(t) = \mu e^{-\mu t}.$$

Během této doby se nemoc může přenášet na vnímavé jedince. Předpokládejme přitom, že nakažený jedinec přímo infikuje v průměru λ vnímavých jedinců za jednotku času. Pak pravděpodobnost p_j, $j = 0, 1, \ldots$, že takové individuum generuje v průběhu své infekční periody právě j nových případů nemoci, je

$$p_j = \int_0^\infty \frac{(\lambda t)^j}{j!} e^{-\lambda t} \mu e^{-\mu t} \mathrm{d}t = \left(\frac{\mu}{\lambda+\mu}\right)\left(\frac{\lambda}{\lambda+\mu}\right)^j.$$

Uvedené vztahy připomínají definici rozdělení pravděpodobností pro nějakou celočíselnou náhodnou veličinu s geometrickým rozdělením. Faktor $\lambda/(\lambda+\mu)$ je možno přitom interpretovat jako pravděpodobnost přenosu nemoci při kontaktu nakaženého jedince s jedincem vnímavým.

Známe-li rozdělení pravděpodobností p_j, potom pro střední hodnotu počtu nových případů generovaných jediným nakaženým individuem dostaneme

$$\mathbf{E}X_1 = \sum_{j=0}^{\infty} j p_j = \frac{\lambda}{\mu}.$$

Z tohoto vztahu vyplývá, že střední hodnota počtu infikovaných jedinců v n-té generaci, kteří pocházejí z jediného nakaženého individua v generaci nulté, je $(\lambda/\mu)^n$. Odtud pro celkovou velikost epidemie N_1 způsobené jedním infikovaným jedincem dostaneme

$$\mathbf{E}N_1 = \sum_{n=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^n = \frac{\mu}{\mu-\lambda} \quad \text{for } \lambda < \mu.$$

Je tedy zřejmé, že střední hodnota velikosti epidemie zůstává konečná tehdy a jen tehdy, když platí $\lambda/\mu < 1$.

Celková velikost epidemie N způsobené K nezávislými infikovanými jedinci je evidentně

$$\mathbf{E}N = \frac{K\mu}{\mu - \lambda} \quad \text{for } \lambda < \mu.$$

6.5.3 Pravděpodobnost konečné extinkce epidemie

K určení pravděpodobnosti extinkce jedné generační linie uvažované epidemie postačí, když vyřešíme rovnici (6.2) s $P(\xi)$ reprezentující vytvořující funkci geometrického rozdělení, tj. rovnici

$$\xi = \frac{\mu}{\lambda + \mu} \sum_{j=0}^{\infty} \left(\frac{\lambda \xi}{\lambda + \mu} \right)^j .$$

Jejím řešením pak dostaneme

$$\xi = \begin{cases} \frac{\mu}{\lambda} & \text{je-li } \lambda \geq \mu, \\ 1 & \text{je-li } \lambda < \mu. \end{cases}$$

Odtud vyplývá pro pravděpodobnost toho, že všech K generačních linií epidemie dříve či později skončí, vztah

$$Pr\{\text{epidemie skončí}\} = \begin{cases} \left(\frac{\mu}{\lambda}\right)^K & \text{je-li } \lambda \geq \mu, \\ 1 & \text{je-li } \lambda < \mu. \end{cases}$$

Z tohoto vztahu je zřejmé, že pokud má epidemie skončit s jistotou, musí být splněna podmínka $\lambda < \mu$. Tato podmínka je ovšem ekvivalentní požadavku, aby střední hodnota počtu nových případů infikovaných přímo jedním nakaženým individuem (θ) byla menší než jedna.

6.5.4 Simulace průběhu malé epidemie

Na závěr stručně popíšeme programový prostředek s názvem EPIDEMIC, určený pro diskrétní simulaci průběhu malé epidemie za podmínky, že jsou splněny předpoklady uvažovaného modelu větvícího se procesu. Tento prostředek vznikl modifikací programu navrženého původně Kindlerem [26] k simulaci šíření lesních požárů.

Jedinci exponované populace jsou na obrazovce reprezentováni znakem '.' a rozmístěni zpočátku v uzlových bodech pravidelné hexagonální mřížky. V následujícím kroku se vysunou ze svých rovnovážných poloh, čímž se vytvoří počáteční "znáhodněná" konfigurace jedinců. Uživatel může modifikovat počáteční konfiguraci následujícími způsoby:

- odstraněním vybraných jedinců,
- prohlášením některých jedinců za mrtvé,
- prohlášením některých jedinců za imunní.

Existují dva základní přístupy k diskrétní simulaci průběhu epidemie:

1. automatická simulace (bez jakýchkoli přerušení v průběhu simulace),
2. simulace po krocích (s přerušením po každé události).

V obou případech uživatel na počátku označí jedince, jenž je zdrojem nákazy. Označený jedinec pak zahajuje přenos nemoci na své sousedy s pravděpodobností rovnou $\lambda/(\lambda+\mu)$. Jeho aktivita však končí v okamžiku, kdy poprvé neuspěje při pokusu nakazit některého ze svých sousedů. Jedinci, kteří se nakazili od zdroje, se stávají infekčními s jistým časovým zpožděním, jehož velikost závisí na jejich vzdálenosti od tohoto zdroje. Každý následně infikovaný jedinec se chová podle týchž pravidel jako importér nemoci.

Na konci každého simulačního experimentu se uživatel rozhoduje, jak v simulaci pokračovat. Při tomto rozhodování má v podstatě dvě možnosti:

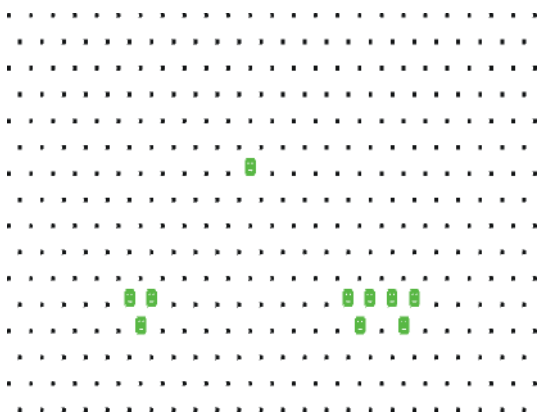
1. zaznamenat výsledek experimentu do souboru a uschovat jej pro případné další použití,
2. modifikovat výsledek experimentu (např. regenerováním mrtvých jedinců nebo zrušením imunity u jedinců imunních) a pokračovat v simulaci.

Vstupní parametry uvedeného programu zahrnují:

- velikost exponované populace,
- průměrnou vzdálenost mezi sousedy,
- střední hodnotu doby trvání nemoci ($1/\mu$),
- počáteční specifickou rychlost přenosu nemoci (λ_0),
- dobu reakce veřejných zdravotnických zařízení (T_r),
- specifickou rychlost přenosu nemoci po zásahu zdravotnických zařízení (λ),
- pravděpodobnost přežití (p_s),
- pravděpodobnost získání imunity (p_i).

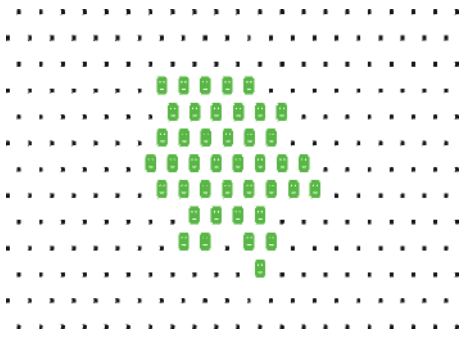
Pro diskrétní simulaci šíření bacilární úplavice byly základní vstupní parametry nastaveny takto: $1/\mu = 7$ dnů, $\lambda_0 = 1 - -5$ den^{-1}, $\lambda = 0.01 - -0.10$ den^{-1}, $T_r = 0$ dnů při aplikaci modelu větvícího se procesu nebo 2 dny jinak, $p_s = 0.999$ a $p_i = 1$ (ekvivalentní předpokladu, že žádné individuum neonemocní víckrát v průběhu téže epidemie). Uvedené nastavení lze považovat za rozumné, protože vychází z dlouhodobé zkušenosti.

Experimentální výsledky prokázaly obecně dobrou shodu s teoretickými předpověďmi. Dva typické výsledky simulačních experimentů jsou uvedeny na obr. 6.9 a obr. 6.10. Jedinci, jež přežili epidemii, mají na obrazovce podobu "smějící se zelené tváře", zatímco ti, kdož zemřeli, jsou označení "červeným křížkem".



Obrázek 6.9: Ilustrace k epidemii bacilární úplavice se 3 nezávislými zdroji

Obr. 6.9 ilustruje situaci po skončení epidemie bacilární úplavice zavlečené do populace nezávisle třemi infikovanými jedinci v případě, že jsou splněny předpoklady modelu větvícího se procesu ($T_r = 0$ dnů, $\lambda = 0.10$ den^{-1}). Epidemie postihla 10 jedinců, což je v souladu s očekávanou velikostí epidemie.



Obrázek 6.10: Ilustrace k epidemii bacilární úplavice s jediným zdrojem

Obr. 6.10 dokumentuje velikost epidemie bacilární úplavice v případě, že je způsobena jediným zdrojem a veřejná zdravotnická zařízení reagují na ni se zpožděním 2 dnů ($\lambda_0 = 2$ den^{-1}, $T_r = 2$ dny, $\lambda = 0.10$ den^{-1}).

Kapitola 7

Programovací prostředky pro simulaci

7.1 Rozbor možností

Jak už jsme poznamenali v odstavci 1.5, v dalším se budeme zabývat jen počítačovou simulací, a tak přívlastek *počítačová* budeme pro stručnost vynechávat. Je zřejmé, že simulační programy (viz úvodní řádky odstavce 1.5) jsou velmi často složité – výpočet podle nich totiž musí být analogií nějakého složitého děje, který chceme poznat a který bývá výslednicí velmi komplexních časoprostorových a jiných vztahů.

Pro překonávání potíží se sestavováním složitých programů existuje pět způsobů, které lze v principu aplikovat i na implementaci simulačních programů:

1. sestavování programů, do nichž zabudujeme předem připravené podprogramy, které se chovají navenek zdánlivě jednoduše a názorně a které při tom provádějí dosti složité výpočty (tedy použití podprogramů jako stavebních kamenů programů);

2. použití předem připraveného „univerzálního" programu", který je řízen daty tak, že některá z nich jeho běh směrují různými cestami;

3. použití programovacích jazyků tak, že texty v nich sestavené a popisující více méně čitelně požadavky na to, co se má počítat, jsou interpretovány speciálním programem, tzv. interpretačním programem neboli interpretem;

4. použití programovacích jazyků tak, že texty v nich sestavené a popisující více méně čitelně požadavky na to, co se má počítat, jsou překládány do odpovídajících programů ve strojovém kódu;

5. použití objektově orientovaných programovacích prostředků jakožto základu pro definice adekvátních problémově orientovaných programovacích prostředků.

Hned na počátku dalšího rozboru uveďme, že bod 3 lze z následujících úvah vypustit, neboť — ačkoliv je interpretace textů historicky chápána jako samostatný způsob řešení potíží programování — jde o zvláštní případ bodu 2, což se v posledních létech stále více prosazuje: interpretace textů u programovacích jazyků interpretačním programem je zvláštním případem zpracování vstupních dat „univerzálním“ programem.

Dnes lze vyloučit z dalších úvah i bod 1, protože přípravu podprogramů vhodných pro pomoc v některých oblastech může lépe nahradit posílání zpráv mezi jednotlivými objekty s tím, že reakce příjemců zpráv jsou adekvátní vyvolání podprogramů, mezi jejichž parametry jsou tito příjemci. Např. vyvolání podprogramu $F(X, Y)$ lze nahradit posláním zprávy objektu X, aby provedl metodu M s parametrem Y, kde M je jen nepatrně přeformulovaná F. V minulosti (v šedesátých a sedmdesátých letech) bylo vyvinuto několik set „balíků podprogramů“, které podporovaly tvorbu počítačových simulačních modelů, z nichž aplikace některých přetrvává někde až dodnes (např. GASP).

Univerzální či spíše univerzálnější „simulátory“, pomáhající podle bodu 2, se uplatňují v různých aplikačních oborech (např. v tvorbě počítačů a ve strojírenské výrobě). Dnes obvykle tyto simulátory reagují na kombinovaná alfanumerická a grafická vstupní data, jejichž rozmanitost je tak veliká, že lze těžko vyslovit nějaké společné zásady pro simulační prostředky sledující zásadu 2; shrnující popis je znemožňován i prudkým vývojem těchto prostředků. U nás jsou z nich poněkud známy např. TAYLOR a WITNESS pro simulaci diskrétních výrob, ve světě však mají podobné prostředky nejrůznější obory lidské činnosti, např. dispečeři energetických sítí, letištní odborníci nebo pracovníci v organizaci zdravotnictví.

Zbývají tedy způsoby 4 a 5. Jim jsou věnovány následující dvě podkapitoly.

7.2 Simulační programovací jazyky

7.2.1 Podstata použití simulačních jazyků

Je vhodné si uvědomit, že programovací jazyky jsou v první řadě realizovány (to jest navrhovány a implementovány) proto, aby uživateli výpočetní techniky pomohly při psaní programů. Pomoc při tom spočívá v tom, že popis pokynů, které řídí práci počítače, má být jednak pro člověka čitelný a jednak ho nemá nutit k tomu, aby zbytečně a po případě i s námahou transformoval své myšlenky.

Na těchto idejích jsou založeny tak zvané **simulační programovací jazyky**, což jsou programovací jazyky, které jsou určeny pro pomoc při sestavování simulačních programů. Než vysvětlíme jejich obecné principy, poznamenejme, že – ve shodě s terminologií ustálenou ve světě – pro ně budeme používat zkráceného názvu **simulační jazyky** (anglicky **simulation languages**), neboť neexistují žádné simulační jazyky, které by nepatřily mezi programovací jazyky.

Když simulujeme nějaký systém, ať už existující (např. orgán konkrétního pacienta) nebo zamýšlený (např. výrobní provoz, který ještě neexistuje), předpokládáme, že takový systém už byl (a to ne jednou) popsán v nějakém jazyku, který není programovací. Může to být např. odborná čeština nebo odborná angličtina, v níž si informace o objektech zájmu své práce sdělují odborníci dané profese. Z tohoto faktu vzniká přirozeně myšlenka adaptovat jazyk dané profese jako programovací jazyk, to jest stanovit jeho přesná syntaktická a sémantická pravidla a tím v prvé řadě odstranit nebezpečí nejasností, které přináší vždy přirozený jazyk, za druhé eliminovat některé komplikace, které slouží u přirozeného jazyka spíše ke kráse než k přesnosti vyjadřování, a za třetí umožnit strojový překlad textů v takto upraveném jazyku přímo do simulačních programů.

Ačkoliv se to vše zdá obtížné, historie ukazuje, že je to možné. Stručně řečeno, přínos simulačních jazyků spočívá v prvé řadě v tom, že chce-li uživatel takového jazyka sestavit simulační program, jehož běh by na počítači realizoval simulační model jistého systému, popíše tento systém v simulačním jazyku, a to podobně jako by ho popisoval svému kolegovi v profesi (snad jen někde se bude vyjadřovat více „po lopatě“), a popis je pak už transformován do příslušného strojového programu automaticky, v dnešní době téměř výhradně kompilací.

Je evidentní, že většinou jde o transformaci velmi složitou, takže realizace kompilátoru (překladače) je obtížná a nákladná. Nadto každý simulační jazyk by měl odpovídat nějakému profesně zaměřenému jazyku. Jelikož však profesních jazyků je mnoho, i simulačních jazyků, a tedy i kompilátorů by mělo být mnoho. V mnoha profesích se však simulace tak vyplatí (a v minulosti vyplatila), že uvedené potíže neodrazují, a historie nás informuje o mnoha stech simulačních jazyků vytvořených počítačovými a softwarovými společnostmi, universitami i aplikujícími (např. výrobními a projekčními) institucemi. (Jistý zlom ve vývoji simulačních jazyků, způsobený v poslední době aplikací objektově orientovaného programování, bude popsán v odstavci 7.2.3.)

Odborník, který pracuje s počítačem, má obvykle jisté schopnosti algoritmizace nebo aplikace matematických metod a je schopen jich použít i při popisu systémů, které studuje. A tak není divu, že se mezi prostředky simulačních jazyků můžeme setkat s běžnými nástroji pro algoritmizaci (s dosazováním

za proměnné nebo častěji za atributy prvků simulovaného systému, s cykly, s větvením, s podmíněnými akcemi, ba i s podprogramy) a s některými matematickými nástroji, které se během doby ustálily jako nástroje popisů dějů v čase (např. s diferenčními a diferenciálními rovnicemi). Je však nutno si uvědomit, že zatímco „konvenční“ (tj. nesimulační) programovací jazyky nutí své uživatele, aby algoritmizaci prováděli do detailů a aby ji chápali jako hlavního nositele pravidel pro každý výpočet, simulační jazyky algoritmizaci nabízejí jako možnost. Když to uživatel považuje za vhodné, může nějaký děj, proces či rozhodnutí formulovat ve tvaru algoritmu, ale ten může „obklopit“ popisy jiného druhu. V popisu simulovaného systému se např. některé z prvků systému mohou chovat více či méně nezávisle, přičemž o synchronizaci algoritmů prováděných několika prvky paralelně se autor popisu nemusí starat.

Příkladem může být popis modelu dopravního systému, v němž se pohybují dopravní prostředky (např. automobily nebo vlaky). Algoritmus chování obecného dopravního prostředku (jeho reakce na signály, na okamžitý stav jeho okolí a na provádění jeho trasy) lze formulovat poměrně jednoduše, zatímco algoritmizovat synchronizaci více takových dopravních prostředků je takřka nad lidské síly; to však provede ve zkompilovaném programu už sám kompilátor. Jako další – podobně ilustrativní příklady – lze uvést boj (pozemní, tankový, letecký či námořní) mezi dvěma nepřátelskými vojenskými útvary, vývoj a dělení nádorových buněk, šíření epidemií či lesních kalamit a automatizovanou výrobu v daném podniku.

Simulačních jazyků bylo implementováno mnoho, přestože implementace je pracná a nákladná. Je to dáno faktem, že prostředky každého profesního přirozeného jazyka – a tedy i každého simulačního jazyka – jsou omezené. Takový jazyk je vhodný pro popis systémů patřících do jisté **sémantické třídy**, kdežto pro ostatní systémy je více či méně nevhodný. Víme, že např. lékař by popsal fyziologii nějakého orgánu těžko v jazyku ekonoma, nebo že pracovník v oboru sociologie by měl podstatné potíže, kdyby měl popsat vývoj dejme tomu zaměstnanosti v jazyku, který používají výrobci lokomotiv. Existují a existovaly snahy vytvořit nějaký „univerzální“ simulační jazyk, jehož sémantická třída by byla poměrně rozsáhlá. Výsledky těchto snah existují a jsou ve světě běžně používány. Šíře jejich sémantických tříd má však za následek, že popis každého simulovaného systému musí být proveden v detailech a „po lopatě“. Je to analogie faktu, že i přirozený jazyk široce profesně orientovaný potřebuje delší popisy objektů zájmu než jazyk orientovaný pro užší profesní skupinu, jejíž členové si v mnohém rozumějí mezi sebou lépe než členové širší profesní skupiny. Např. v nukleární terapii nádorů je terminologie, které rozumějí odborníci v této profesi; příslušné termíny by museli vysvětlit např. jiným lékařům, s nimiž by v dané problematice přišli do styku, ale na základě obecné odborné terminologie je takové vysvětlení možné, zatímco definovat termíny

nukleární nádorové terapie způsobem přijatelným dejme tomu ekonomovi nebo odborníku v hutnictví oceli je takřka nemožný (a ovšem i nepraktický) cíl.

Sémantická třída žádného z „univerzálních“ simulačních jazyků nepokrývá všechny možné dynamické systémy, takže přívlastek univerzální je třeba chápat metaforicky (proto jej dáváme do uvozovek). Ačkoliv se zdá, že „univerzální“ simulační jazyky v sobě spojují nevýhody jak ostatních simulačních jazyků (faktická neuniverzálnost), tak konvenčních algoritmických jazyků (nutnost podrobného popisu každého simulovaného systému), je jejich využití ve světě dosti rozšířeno. Je to jednak proto, že stále přibývají nové obory, které služby simulace chtějí využívat, a jednak proto, že komplikace v popisech systémů, jak to „univerzální“ simulační jazyky vyžadují, nejsou zásadní. Popis je stále mnohem jednoduší a čitelnější než v konvenčních programovacích jazycích.

7.2.2 Základní klasifikace simulačních jazyků

Už v první kapitole jsme se zmínili, že v systémech existují obecně transakce a permanentní prvky čili aktivity. Systémy, které obsahují prvky obou těchto kategorií, budeme nazývat **systémy typu AT** (zkratka Aktivity-Transakce) nebo zkráceně **AT-systémy**. Lze si představit systémy tak proměnné v čase, že v nich žádné aktivity nejsou, tj. že všechny jejich prvky chápeme jako transakce. Tyto systémy budeme nazývat **systémy typu T** čili **T-systémy**. Existuje však i opačný extrém, **systémy typu A** čili **A-systémy**, v nichž nejsou žádné transakce. V takových systémech zanedbáváme v podstatě všechny strukturní změny a celý děj, který v nich v čase probíhá, abstrahujeme do nějakých změn hodnot atributů (většinou jde o atributy numerické, resp. booleovské).

Nechť L je simulační jazyk a C jeho sémantická třída. Když C obsahuje pouze A-systémy, řekneme, že L je **jazykem základního typu A**. Když C obsahuje pouze T-systémy, řekneme, že L je **jazykem základního typu T**. V ostatních případech řekneme, že L je **jazykem základního typu AT**.

Upozorňujeme, že sémantická třída jazyka základního typu AT nemusí být omezena na AT-systémy. Existují sice jazyky základního typu AT, v nichž lze popsat pouze systémy obsahující aspoň jednu aktivitu a aspoň jednu transakci, avšak obecně platí, že sémantická třída jazyků základního typu AT může obsahovat i systémy typu A (což jsou v mnoha případech degenerované případy systémů typu AT) a – v případech mnohem méně častých – i systémy typu T.

7.2.3 Jazyky základního typu A

Sémantické třídy těchto jazyků jsou dosti omezeny, a tak lze tyto jazyky charakterizovat velmi prostě: až na několik (v zásadě bezvýznamných) výjimek

jde o následující tři skupiny.

7.2.3.1 Jazyky pro spojitou simulaci

Tak zvané **jazyky pro spojitou simulaci** (anglicky **continuous systems simulation languages**) jsou simulační jazyky, v jejichž sémantických třídách jsou spojité systémy, které lze popsat pomocí obyčejných diferenciálních rovnic, v nichž se vyskytují derivace atributů jednotlivých prvků podle času. Prostředky těchto jazyků vycházejí obvykle z ustálené matematické symboliky: rovnost časové derivace atributu x pravé straně $f(x, \ldots)$ může uživatel mnoha takových jazyků – zejména amerických – přepsat na rovnost hodnoty atributu x časovému integrálu pravé strany, přičemž symbol integrálu se nahrazuje v různých jazycích různými alfabetickými výrazy jako INT, INTEG, INTEGRAL apod. Místo $x' = f(x, \ldots)$ se tedy musí uvést $x = INTEG(f(x), \ldots)$, dt se přitom vynechává. Starší (avšak zejména v USA stále používané) jazyky pro spojitou simulaci vedou jejich uživatele k tomu, aby simulovaný systém popsali jako konkrétní zapojení idealizovaného analogového počítače, tj. systému složeného z analogových sčítaček, násobiček, integrátorů a dalších prvků, jejichž nabídka fakticky charakterizuje úroveň takového jazyka.

7.2.3.2 Jazyky pro simulaci technického vybavení počítačů

Tak zvané **jazyky pro simulaci technického vybavení počítačů** (anglicky **hardware simulation languages**, někdy – ne zcela logicky – nazývané **hardware description languages**, tj. jazyky pro popis technického vybavení počítačů) jsou simulační jazyky, které umožňují svým uživatelům popsat strukturu počítače nebo jeho části v termínech, jež jsou obvyklé mezi počítačovými techniky. Tyto jazyky lze klasifikovat do tří podskupin, a to podle toho, na které úrovni umožňují popis (a tedy i simulaci) počítače nebo jeho části:

1. **Jazyky na úrovni obvodů**. Uživatel popíše procesor nebo jeho část tak, jak by ji popsal specialistovi v elektrotechnice, tj. jako síť tranzistorů, kondenzátorů, odporů a dalších prvků. Reakce počítače spočívá ve spojité simulaci elektronických dějů v takovém systému. Je evidentní, že jazyky tohoto typu lze s výhodou aplikovat i na simulaci elektronických obvodů, které nemají s výpočetní technikou nic společného; přesto se však ve světové literatuře zahrnují – byť nepřesně – mezi jazyky pro simulaci technického vybavení počítačů.

2. **Jazyky na úrovni logických hradel**. Uživatel popíše procesor nebo jeho část jako síť složenou z „logických prvků“, v níž se pohybují „bo-

oleovské“ signály (tedy logická nula a logická jednička). Logické prvky reagují na své vstupní signály způsobem, který lze popsat pomocí logických operací známých např. z matematické logiky (logické **and** čili a, **or** čili nebo, **not** čili ne apod.). Logické prvky někdy mohou své výstupní hodnoty v čase zpozdit. Reakce počítače na takový popis je diskrétní simulace děje v popsané síti.

3. **Jazyky na úrovni registrů a mikroprogramování**, nazývané v angličtině **register transfer languages**. Uživateli dávají k dispozici knihovnu standardních prvků odpovídající takovým složkám počítače, jako jsou registry různých druhů a další paměťová místa, sčítačky, střádače, násobičky atd. Reakce počítače na takový popis je diskrétní simulace dějů v popsaném systému.

7.2.3.3 Jazyky pro simulaci transportních procesů pomocí tzv. systémové dynamiky

Tato skupina simulačních jazyků nemá speciální název ve světě běžně používaný, protože takových jazyků je velmi málo. Mají však důležitou roli ve spojitém modelování ekonomických toků (toků financí nebo materiálu) a – přibližně od konce 80. let – i procesů v životním prostředí. Jejich uživatel popíše studovaný systém jako soustavu „nádob“ spojených „kanály“, jimiž se mezi nádobami pohybuje „látka“; ta může odpovídat chemické látce nebo množství jedinců v systémech životního prostředí nebo financím či množství obchodovaného produktu v ekonomii. Rychlosti v kanálech mohou být vyjádřeny pomocí výrazů, v nichž se vyskytují objemy „nádob“, takže tyto rychlosti a tím i objemy se mohou v čase měnit dosti složitým způsobem. Uvedená metoda popisu byla koncem 50. let navržena J. Forresterem, který ji s úspěchem nabídl ekonomům, a ti občas píší o forresterovských modelech a jazycích. Podle nejznámějšího z jazyků této skupiny, který nese název DYNAMO, se občas setkáme i s termínem „jazyky typu DYNAMO“.

Pomocí uvedených jazyků základního typu A bylo možno nejprve simulovat systémy o desítkách rovnic, resp. prvků, avšak během doby – jak se zvyšovala rychlost a paměťová kapacita počítačů – se rozsah simulovaných systémů zvětšoval. Možnosti dnešních počítačů nevylučují simulovat systémy, které mají mnoho set prvků, avšak v takovém případě se naráží na hranice lidské psychiky. Jedinec není schopen popsat bez chyb tak velký systém prvek za prvkem. A proto takřka všechny moderní jazyky základního typu A umožňují zavádět tzv. **makra**, což jsou jakési podsystémy, které lze kopírovat a které uživatel může definovat (nebo převzít z knihovny maker) a použít je stejně jako ostatní – jednodušší – bloky, resp. rovnice. Např. v jazyku pro spojitou simulaci může uživatel definovat makro „chemický reaktor“ jako systém slo-

žený z mnoha (analogových) sčítaček, integrátorů a dalších prvků, podobně může definovat makra „výměník tepla“, „rektifikační kolona“ a další a pak formulovat, že jím popsaný simulovaný systém chemické výroby je síť složená z takových a takových chemických reaktorů, výměníků tepla, rektifikačních kolon a dalších „technologických“ prvků, tak a tak propojených. Kompilátor pak „nakopíruje“ do paměti pro každé makro jeho vnitřní strukturu podle toho, jak bylo dané makro definováno, takže např. každý chemický reaktor má „své“ sčítačky, integrátory a další prvky, které mu dle definice příslušejí.

Pro výše zavedené názvy skupin jazyků základního typu A jsme vycházeli z toho, že jsou mezinárodně přijaté. Musíme však upozornit, že ani jeden z nich není zcela výstižný. Tak např. jazyky pro simulaci elektronických obvodů by měly logicky patřit mezi jazyky pro spojitou simulaci, ale odborná veřejnost to tak nebere. Nadto téměř všechny dnes používané jazyky pro spojitou simulaci mohou připouštět nespojitosti v chování simulovaného systému; příkladem je blok nazývaný *signum* (znaménko), který má jeden vstup a na svém výstupu dá jednu z hodnot 1 a -1, a to podle toho, zda je číslo na vstupu nezáporné nebo záporné. Zatímco výstup ze sčítačky se v čase mění spojitě, pokud se takto mění i hodnoty na jejím vstupu (a výstup z integrátoru se mění spojitě dokonce i tehdy, když integrovaná hodnota na jeho vstupu se mění skokem), u prvku signum se změní hodnota na jeho výstupu skokem, když se hodnoty na jeho vstupu mění spojitě od kladných na záporné nebo naopak.

Existují spojité systémy, které nelze popsat pomocí obyčejných diferenciálních rovnic. Platí to např. pro tak zvané **systémy s rozdělenými parametry**, při jejichž popisu musíme použít parciálních diferenciálních rovnic. Tak vzniká otázka, kam zařadit simulační jazyk, který umožňuje popsat systémy s rozdělenými parametry. Světová odborná veřejnost se na jednotné odpovědi dosud neshodla, avšak poznamenejme hned, že to zatím nevadí. S numerickými metodami, které by bylo možno algoritmizovat pro řešení mnoha parciálních diferenciálních rovnic, jsou tak velké potíže, že – zatímco jazyků na úrovni obvodů je několik set – lze simulační jazyky orientované na systémy s rozdělenými parametry spočítat na prstech; v klasifikaci se o nich už dále nebudeme šířit.

Na závěr je nutno ještě upozornit, že logicky by mezi jazyky pro simulaci technického vybavení počítačů (viz odstavec 7.2.3.2) měly patřit i jazyky pro simulaci počítačových sítí a složitějších systémů složených z procesorů. Takové jazyky sice existují, ale to už nejsou jazyky základního typu A, neboť v jejich sémantické třídě existují např. počítačové sítě, které si předávají úlohy a protokoly, což jsou transakce: mají totiž své atributy, které během jejich existence informují o jejich charakteru a o tom, jak budou na příklad interagovat s tím či oním prvkem sítě.

7.2.4 Jazyky základního typu AT

Vyjděme z příkladu počítačových sítí, kterým byl ukončen předcházející odstavec. Počítačovou síť zpravidla simulujeme tak, že předpokládáme pevnou strukturu jejích počítačů, terminálů a spojů mezi nimi, avšak připouštíme, že v síti těchto permanentních prvků se „pohybují“ úlohy, že tyto úlohy do sítě vstupují a po čase z ní mizejí. Jsou to tedy transakce.

Místo počítačové sítě si můžeme představit železniční síť, síť pouliční dopravy ve městě, letištní plochu, velkou nádražní nebo letištní budovu, výrobní nebo montážní systém, vše má permanentní prvky a transakce. Jde o tak zvané **systémy hromadné obsluhy**, tj. systémy, v nichž se tvoří fronty – většinou uspořádané množiny transakcí.

Když chceme nějaký systém simulovat, musíme ho více méně přesně v simulačním programu popsat, tj. musíme si uvědomit jisté vlastnosti systémů, které zamýšlíme simulovat. Odborníci si brzy (koncem 50. let) povšimli, že pro systémy hromadné obsluhy hrají podstatnou roli děje spojené s jednotlivými transakcemi nebo aktivitami. To, co se v systému hromadné obsluhy děje, je výslednicí toho, jak se v systému „pohybují“ transakce nebo jak v něm aktivity „pohybují“ transakcemi.

Příkladem může být vlak v železniční síti. Lze na něj jako na transakci pohlížet tak, že má svou trasu (seznam míst, kterými má projet) a podle ní provádí cyklus odpovídající přesunům z jednoho místa do místa následujícího. Každý krok tohoto cyklu obsahuje vlastní přesun, stání ve stanicích a případné stání před semafory. To, zda je nutno zastavit před semaforem, je dáno testem, zda je na semaforu „zelená“, tj. na to, jaká je hodnota jistého atributu aktivity, jíž semafor je. (Poznamenejme, že v tomto příkladě lze takový atribut chápat jako booleovský, nazvaný dejme tomu *volno*, jehož hodnota je buď *ano* (vlak může jet) nebo *ne* (vlak musí čekat).) U každého vlaku pak můžeme pomocí simulace určit, jak mnoho času prostojí před semafory, a tak simulovaný systém popíšeme tak, že vlak má numerický atribut *čas ztrávený čekáním před semafory*, k jehož hodnotě se na konci každého čekání před semaforem přičte délku tohoto čekání.

Na základě tohoto jednoduchého příkladu poznáváme, že typický „děj“ vlaku lze popsat algoritmicky podobně jako např. podprogram; i v ději vlaku jsou dosazení (za atributy), větvení, resp. podmíněné příkazy (test na „zelenou“ semaforu a následující čekání nebo vypuštění tohoto čekání) a – jak už jsme naznačili – cykly. Je evidentní, že v jiných případech simulovaných systémů mohou být algoritmy dějů transakcí mnohem složitější a tvůrce simulačních programů přitom s chutí použije i jiné prostředky algoritmizace jako např. podprogramy.

Zůstaňme ještě u příkladu s vlaky. Děj v železničním systému můžeme po-

psat také jiným způsobem. Spojíme jeho akce s aktivitami (stanicemi, semafory, kolejovými úseky atd.), jako kdyby je tyto prvky aktivně vykonávaly a „pohrávaly“ si při tom s transakcemi - vlaky. Vlaky jako by byly pasivními prvky, bez vlastního děje: ani hodnoty svých atributů si nemění samy, ty jim mění aktivity, když si s nimi „pohrávají“.

Tento objev lze rozšířit i na systémy, kde nevznikají fronty. Např. život buňky lze popsat algoritmem, jak přechází z jedněch fází do dalších, přičemž se v jedné z nich může rozdělit. Buňky jsou transakce a např. jejich stavy (nebo mnohem složitější biologické entity související nejen s buněčným stavem, ale i s jejím umístěním apod.) jsou aktivity.

Odborníci si toho všeho brzy všimli a využili svých pozorování k návrhu simulačních jazyků základního typu AT. Vznikly dvě kategorie těchto jazyků, které budeme nazývat **jazyky typu AT** (nebo stručně **AT-jazyky**) a **jazyky typu TA** (stručně **TA-jazyky**). Slovo „základní“ zde tedy vynecháváme.

Jazyky typu AT umožňují svým uživatelům popsat systém typu AT tak, že se popíše „skelet“ aktivit a třídy transakcí. Pro každou třídu transakcí se uvede, jaké mají její transakce atributy a dále „algoritmus životních pravidel“, jímž se každá její transakce řídí. Jazyky typu TA umožňují svým uživatelům popsat systém typu AT tak, že popíší aktivity spolu s „algoritmy jejich životních pravidel“, ovšem s tím, že mezi takovými aktivitami mohou být „generátory transakcí“, které vytvářejí transakce a posílají je k jiným aktivitám.

Prvky, s nimiž jsou spojena životní pravidla, lze nazvat **aktivními prvky** a ostatní **prvky pasivními**. Jak je patrno z právě zavedených názvů, využíváme při identifikaci typu (nikoliv základního typu!) simulačního jazyka jejího posledního místa k určení kategorie aktivních prvků; např. AT říká, že děj nesou transakce. Tuto konvenci budeme aplikovat i později.

Popis simulovaného systému v některém z jazyků typu AT či TA je kompilátorem jazyka zpracován tak, jak to odpovídá skutečné existenci v čase: algoritmy prováděné jednotlivými prvky, tj. životní pravidla, jsou „synchronizovány“ ve společném čase (řečeno neodborně, jejich provádění je promícháno). Jako ilustraci si můžeme představit to, že když na nějakém úseku železniční sítě se pohybuje vlak V, na jiném úseku se může rozjet jiný vlak, který může vlaku V zablokovat anebo naopak odblokovat vjezd do stanice či příjezd k semaforu. Tato automatická synchronizace je jedním z největších přínosů, které jazyky typu AT a jazyky typu TA poskytují.

Vliv společného času, v němž by měly prvky existovat, a s ním spojená „synchronizace“ tvoří důležitou odlišnost životních pravidel od běžných algoritmů. Když např. popisujeme životní pravidla vlaku, musíme v nich uvést, že některé jejich kroky zaberou nějaký čas. Kdyby čas neměl význam, nebylo by možné ani synchronizovat to, jak jsou různými prvky simulovaného systémy prováděna jejich životní pravidla. Jazyky typu AT i jazyky typu TA řeší pro-

blém nenulového trvání jednotlivých kroků životních pravidel pomocí jakýchsi přerušení těchto pravidel: např. tvrzení, že vlak se pohybuje z místa A do místa B spojitě a nějakou dobu mu to trvá, popíše uživatel těchto jazyků obvykle tak, že vlak je v místě A, pak nějakou dobu „nedělá nic“ (provádění svých životních pravidel přeruší) a pak se najednou octne v místě B. Krok, kterým vlak přeruší provádění svých životních pravidel, se nazývá **plánovací příkaz** (anglicky **scheduling statement**).

Rozeznávají se dva hlavní druhy plánovacích příkazů: **imperativní** čili rozkazovací příkaz (angl. **imperative statement**), který bychom mohli vyjádřit slovy „čekej po dobu tak a tak dlouhou“, a **interrogativní** čili tázací příkaz (angl. **interrogative statement**), který lze přiblížit slovy „čekej, až bude splněna daná podmínka“. Trvání přesunu vlaku mezi dvěma místy se evidentně vyjádří imperativním příkazem („čekej po dobu nutnou k přesunu“), kdežto čekání na uvolnění dráhy změnou signálu vysílaného semaforem vyjádříme interrogativním příkazem („čekej, až se změní barva na semaforu“).

Promyslíme-li podrobněji to, co bylo právě o obou druzích jazyků základního typu AT řečeno, napadne nás, že by snad bylo vhodné, kdyby existovaly simulační jazyky, které umožní některé děje spojit s transakcemi a jiné s aktivitami. Je to jednak obecnější, než děj nutně vázat na jeden pevně daný druh prvků, a jednak se o tuto možnost uchází i realita kolem nás. Popisujeme-li železniční síť, bude se nám zdát přirozené spojit jízdu vlaku s vlakem (tedy s transakcí), ale zablokování trati semaforem bychom v nejjednodušší abstrakci nejraději spojili právě se semaforem (tedy s aktivitou). Když místem se semaforem projede vlak, semafor trať zablokuje pro další vlaky, a to na jistou dobu, která na nich nezávisí. (V jednoduché abstrakci můžeme železniční síť popsat tak, že semafor se sám po propuštění vlaku „překlopí na červenou“, pak jistou dobu „čeká“ a pak se „sám překlopí na zelenou“.

Zkušenost v celém světě však ukázala, že nabízet uživatelům „smíšené“ jazyky typu AT a TA nepůsobí ani tak pohodlí jako chyby v popisu. Potíž je totiž v tom, že v reálných systémech nastávají často interakce mezi transakcí a aktivitou, u nichž nemáme chuť komplikovat si život s rozhodnutím, zda je máme spojit s transakcí nebo s aktivitou. Když např. nákladní vlak přijede na místo, kde má být naložen, můžeme říci, že on sám si aktivně vyžádá naložení a dovést tuto představu do extrému tím, že formulujeme, že vlak si sám vezme příslušný materiál. Ale stejně můžeme formulovat, že vlak se chová v místě nakládky zcela pasivně, zatímco místo samo mu dodá ze svých „zásob“ příslušný díl. U složitých systémů se pak snadno může stát, že jednu a tutéž interakci bychom omylem formulovali mezi životními pravidly transakce i aktivity, takže by se v simulačním modelu prováděla dvakrát častěji než v simulovaném systému.

Z tohoto důvodu autoři jazyků základního typu neztrácejí pracovní síly na

návrzích a implementacích smíšených jazyků typů AT a současně TA; až na nepatrné výjimky raději nabídnou uživateli jazyk orientovaný výhradně na typ buď AT nebo TA, aby ho nevedl k právě uvedeným omylům. Těm, kdo chtějí být za každou cenu svobodní vůči volbě, s kterými prvky spojovat děj, mohou pomoci jazyky základního typu T, o nichž budeme blíže informovat v oddíle 7.2.5.

Poznamenejme, že problémy, jako je třeba výše zmíněná „samostatnost“ semaforu, lze snadno překlenout např. zavedením fiktivní transakce, která jako by „seděla“ na aktivitě - semaforu a tam se starala o přepínaní jeho propustnosti. Jde o jakousi „degenerovanou“ transakci, která „nevyužije“ toho, že se může v síti aktivit pohybovat a dokonce může systém opustit. Zatímco transakce mohou takto degenerovat na aktivity, není možno žádným způsobem modifikovat prostředky pro formulaci aktivit k zavedení transakcí.

Systém, který simulujeme, lze málokdy popsat zcela přesně. Když něco přesně neznáme, prohlásíme to za neurčité či náhodné. V souhlase s tím nabízejí simulační jazyky generátory pseudonáhodných čísel, kterými uživatel svou nejistotu a své neznalosti nahradí. Když např. nemůžeme přesně formulovat, jaké požadavky budou na počítačovou síť přicházet a v kterých okamžicích to nastane, nahradíme vše „náhodnými“ dobami příchodů požadavků a „náhodnými“ parametry těchto požadavků. Tento přístup se uplatňuje zejména při výzkumu systémů hromadné obsluhy, a tak se generátory pseudonáhodných čísel vyskytují téměř ve všech jazycích základního typu AT (ba i v jiných programovacích jazycích). O těchto generátorech bylo již podrobněji pojednáno v těchto skriptech (v odstavci 4.4).

Mezi jazyky typu AT se proslavil GPSS [44] (v některých svých verzích interpretovaný jako General Purpose System Simulator, v jiných jako General Programming Simulation System), který byl navržen ve Spojených Státech už na začátku 60. let a od té doby je v této zemi i v Evropě používán až do dnešních dnů. Z jazyků typu TA vynikly ve své době navzájem dosti odlišné jazyky nazývané GERT (Graphical Evaluation and Review Technique), které se sice v dnešní době už velké oblibě netěší, ale které přešly do některých simulačních prostředků jako složky „univerzálních“ simulačních programů, o nichž jsme se zmínili v bodě 2 odstavce 7.1. Z naší země pochází moderní simulační prostředek typu TA nazvaný PMF (Project Management Forecast) [49].

7.2.5 Jazyky základního typu T

V exaktních oborech je běžné, že něco, co neexistuje, se chápe jako cosi, co by mohlo existovat, ale „má to nějaký nulový parametr“. Místo abychom řekli, že měření nebo výpočet byl bez chyby, řekneme, že jeho chyba je nulová. Místo toho, abychom řekli, že nějaký fyzický objekt není hmotný, resp. nenese elek-

trický náboj, řekneme, že hmotnost, resp. elektrický náboj, objektu jsou nulové. Podobný přístup lze aplikovat i pro aktivity: lze je chápat jako transakce, které „nevyužívají“ možnost měnit svou vzájemnou konfiguraci, vstupovat do systému po začátku jeho existence a opouštět jej před koncem jeho existence. Jelikož systém je abstrakce definovaná na realitě, je takový přístup oprávněný: v takové abstrakci prostě abstrahujeme od toho, že víme o vzájemné neměnnosti některých prvků daného systému.

Je vhodné si uvědomit, že myšlenkovou cestou, kterou jsme právě naznačili, se dostáváme velmi jednoduše ke konkrétním T-systémům. Vidíme, že pro T-systémy není nutno studovat a modelovat nějaké těžko zvládnutelné objekty, v nichž „se vše mění“.

Uvedené pojetí bylo realizováno v roce 1964 v simulačním jazyku SIMULA, který byl později nazýván SIMULA I [9]. Nebyl to ještě známý objektově orientovaný jazyk (nazývaný SIMULA 67), jehož principy byly sděleny světové odborné veřejnosti až roku 1967, nýbrž vskutku jazyk zaměřený na simulaci (diskrétních) systémů. Jeho autoři jej charakterizovali jako jazyk pro programování a popis systémů s diskrétními událostmi (a language for programming and description of discrete event systems). Jeho uživatel byl vskutku veden k tomu, aby v případě, že by chtěl popsat nějaký systém typu AT, chápal aktivity jako degenerované transakce, u nichž není využito možností měnit jejich vzájemnou konfiguraci, vstupovat do systému po začátku jeho existence, opouštět je před koncem jeho existence a dokonce mít životní pravidla.

K těmto jazykům typu T se přibližují jazyky pro objektově orientované programování. K tomuto fenoménu se však vrátíme ještě v následujících odstavcích této kapitoly.

7.2.6 Jazyky s elementárními prvky

Jak už jsme poznamenali, mají životní pravidla mnoho analogií s algoritmy. Tento fakt podněcuje tři ideje. Ta první, mohli bychom říci skromnější, spočívá ve snaze použít k popisu životních pravidel zvyklostí (včetně konkrétních syntaktických pravidel) známých a ustálených v konvenčních algoritmických jazycích. Uživatel simulačního jazyka by se nemusel učit pravidla, jak zapisovat dosazení, cykly, větvení a další vyjadřovací algoritmické prostředky, nýbrž by aplikoval své znalosti a dovednosti z jiného jazyka (v úvahu přicházely postupně FORTRAN, ALGOL, BASIC, C a JAVA). Druhá, méně skromná idea spočívá v tom, že by se životní pravidla (a vlastně celé popisy simulovaného systému, tedy celé texty v simulačním jazyku) mohly automaticky překládat do textů v nějakém konvenčním algoritmickém jazyku: algoritmické prostředky by se prostě okopírovaly a kompilátor simulačního jazyka by se podstatně zjednodušil. A třetí, nejnáročnější idea spočívá v tom, že by vůbec nemusel existo-

vat nějaký překladač či kompilátor, kdyby se algoritmické struktury nějakého rozšířeného programovacího jazyka doplnily vhodnými podprogramy pro generování a likvidování transakcí a pro plánovací příkazy.

První idea, izolovaná od druhé a třetí, nevyžaduje nějaké zvláštní triky při implementaci simulačního jazyka, avšak nemá sama ani nějaký podstatný dopad. Např. algoritmické prostředky jazyků GPSS a GERT, o nichž jsme se zmínili v závěru předcházejícího odstavce, jsou daleko od všech zvyklostí známých z konvenčních algoritmických jazyků.

Pokud chceme aplikovat druhou, resp. třetí, ideu, je otázka, jak to udělat. Konkrétně vzato, u druhé ideje je třeba odpovědět na otázku, jakou formu bude mít text vzniklý překladem životních pravidel třídy prvků. A u třetí ideje musíme vědět, čemu by měly odpovídat v oblasti konvenčních algoritmických jazyků životní pravidla.

Odpověď se zdá být prostá: životní pravidla formulovaná pro třídu prvků jsou prováděna postupně všemi jejími instancemi, a tak by bylo vhodné chápat životní pravidla jako podprogram (nebo je jako celek do programu překládat): podprogram je přece také cosi, co se formuluje jednou, ale je to vykonáváno mnohokrát.

Naneštěstí taková odpověď je nepřesná a nepoužitelná. Potíž je v „přepínání“ mezi životními pravidly, jak jsme se o tom zmínili v souvislosti s plánovacími příkazy: něco podobného se mezi podprogramy neděje. Ukázalo se, že jediný přístup, jak životní pravidla více či méně převést na texty v konvenčních algoritmických jazycích, spočívá v rozdělení životních pravidel na nepřerušitelné, to jest okamžité akce a ty formulovat jako podprogramy. Nechceme zde zabíhat do detailů, a tak uvedeme jen několik postřehů.

Životní pravidla mají tvar $A_1, P_1, A_2, P_2, \ldots, A_n, P_n, A_{n+1}$, kde A_i jsou úseky, které lze vyjádřit pomocí prostředků konvenčních algoritmických jazyků, a P_i jsou plánovací příkazy, dejme tomu tvaru „čekej po dobu T_i“ nebo „čekej, až bude splněna podmínka B_i“. Definujeme v systému $n+1$ tříd $E_1, E_2, \ldots, E_n, E_{n+1}$ jakýchsi fiktivních prvků – budeme jim říkat **elementární prvky** – které jsme původně v systému nepozorovali. Každý z těchto elementárních prvků má jeden referenční atribut R a s třídou E_i jsou spojena pravidla A_i, za nimiž následuje ještě doplněk D_i, který má tvar „generuj“ prvek třídy E_{i+1}, okopíruj na jeho atribut R současnou hodnotu atributu R a aktivuj tento prvek za čas T_i, resp. až bude splněna podmínka B_i. Místo abychom k dané třídě formulovali životní pravidla

$$A_1, P_1, A_2, P_2, \ldots, A_n, P_n, A_{n+1},$$

necháme ji pasivní (bez životních pravidel). Když je však generován prvek X této třídy, spolu s ním necháme generovat i prvek třídy E_1 a za jeho atribut R mu dosadíme X. Když si v duchu probereme, co se bude dít, zjistíme, že

prvek X bude „ovládán“ postupně prvky $A_1, A_2, \ldots, A_n, A_{n+1}$, které se budou postupně vytvářet, předávat si „ovládání“ prvku X pomocí atributu R a čekat při krocích tohoto ovládání zcela stejně, jako by tomu bylo dle původně zamýšlených životních pravidel.

Životní pravidla elementárních prvků nejsou přerušována plánovacími příkazy, a tak splňují to nejdůležitější: elementární prvky lze bez komplikací převést na podprogramy, ba je možno vše zařídit tak, že přímo – tedy bez překladu – se elementární prvky mohou formulovat jako podprogramy. Lze tedy realizovat druhou ba dokonce i třetí ideu. Avšak zavádění elementárních, tedy fiktivních, prvků je nepřirozené a „rozbíjení“ původně zamýšlených celistvých životních pravidel mezi elementární prvky je komplikované, stejně jako explicitní naprogramování toho, aby si tyto prvky v daných časových relacích „předávaly vládu“ nad jedním a týmž „nefiktivním“ prvkem simulovaného systému.

Výhody – jednoduchost kompilátoru nebo dokonce jeho úplné odstranění – vedly v 60. létech k jisté oblibě jazyků s elementárními prvky. Patřily sem populární simulační jazyky SIMSCRIPT I a II (vyžadující kompilátory) a GASP. Postupem doby však převládl v lidské civilizaci nesouhlas s obtížemi při programování, takže aplikace takových jazyků takřka vymizela. V 90. létech nastala jistá renezance, a to ve vztahu k objektově orientovaném programování (viz následující odstavec).

Až na výjimky jsou elementárními prvky transakce. Když je symbolizujeme výrazem T_E, můžeme říci, že právě zmíněné jazyky jsou typu ATT_E a patří mezi jazyky základního typu AT. V americké literatuře se elementární prvky nazývají **events** (česky **události**), avšak tento termín je – i v angličtině – používán i v jiných souvislostech (mimo jiné i v oboru jazyků typu TA a typu AT – tedy jazyků bez elementárních prvků – pro nepřerušené úseky výpočtů, tedy pro úseky, které jsme výše značili jako $A_1, A_2, \ldots, A_{n+1}$), a tak ho nedoporučujeme v uvedeném významu používat. Odstraní se tím mnoho zbytečných nedorozumění.

Souhrn prostředků pro deterministické přepínání výpočtu mezi několika algoritmy se nazývá **kvaziparalelní systém** a programování s použitím kvaziparalelních systémů se nazývá **kvaziparalelním programováním**. Zdůrazňujeme, že jde o deterministické přepínání, tj. takové, které nezávisí na okamžitém stavu počítače. (Nedeterministické přepínání známe např. z multiprogramování a multitaskingu, jak jsou zabudovány v některých operačních systémech.) Jazyky typu AT a typu TA tedy mají zabudován kvaziparalelní systém, zatímco použití jazyků s elementárními prvky je charakteristické tam, kde stavíme na základě (konvenčního) programovacího jazyka, který kvaziparalelní programování neumožňuje. Běžně oblíbené programovací jazyky (FORTRAN, Pascal, Basic, LISP, AmallTalk, JAVA, C ani C++) kvaziparalelní

programování neumožňují.

Detaily týkající se klasifikace simlačních jazyků a příklady některých konkrétních jazyků lze nalézt v monografii [22].

7.3 Jazyky pro objektově orientované programování

7.3.1 Definice objektově orientovaného programování

Ve shodě s názory přijatými dnes už drtivou většinou odborníků v celém světě můžeme vymezit pojem **objektově orientovaného programování** (v dalším OOP) jako proces tvorby počítačových programů s pomocí reprezentace znalostí ve formě tříd, to jest ve formě abstraktních entit, pro které platí následující tvrzení.

1. Třída obsahuje pojmenované (identifikovatelné) hodnoty čili **atributy** a algoritmy čili **metody**.

2. Třída je „prototypem“ libovolného, předem neurčeného množství konkrétních entit, tak zvaných **instancí třídy**. Místo o instanci se často mluví o objektu. Když někdo řekne, že je něco **objekt**, chce tím obvykle vyjádřit, že je to instance nějaké – snad nedůležité či v daném kontextu neznámé – třídy. O instanci, která je vytvořena z nějaké třídy C, říkáme, že je to **instance třídy** C nebo že **patří do** C.

3. Každá instance třídy dejme tomu C nese své vlastní atributy zavedené podle jejich definice v třídě C a na požádání od (jiných) instancí téže třídy nebo jiných tříd (tedy od jiných objektů) může provést jakoukoliv metodu v třídě formulovanou, pokud provádění takové metody nebylo v definici třídy C zablokováno (viz bod 5). Když nějaký objekt požádá instanci X aby provedla metodu F, říkáme, že **poslal zprávu** instanci X se **selektorem** F a že X je **adresátem** této zprávy. Anglicky se zpráva nazývá **message** a poslání zprávy se nazývá **message transfer**.

4. U každé třídy lze stanovit „privátní“ čili **chráněné** atributy: tato vlastnost se přenáší na atributy každé instance této třídy tak, že s nimi může manipulovat ona sama instance, když dostane zprávu, aby provedla nějakou metodu.

5. Stejně jako jsou chráněny některé atributy, mohou být chráněny i některé metody; ty pak může aplikovat instance, k níž jsou vztaženy, jen jako součást provádění jiných - případně i nechráněných - metod. Jinými slovy,

chráněné metody mohou vystupovat pouze jako selektory zpráv, které posílá instance sama sobě.

6. Každá třída může být obsahově obohacena čili **specializována** přidáním nových atributů a formulováním nových metod. Takto specializovaná třída se stává novou třídou, tzv. **podtřídou** výchozí třídy.

7. Při formulaci třídy není nutno nic vědět o jejích případných podtřídách, ani o jejich případném počtu.

8. Některé metody v třídě lze prohlásit za **virtuální**. Virtuální metoda je ta, o níž v definici třídy stanovíme, že má smysl, ale nemusíme stanovit, jaký tento smysl konkrétně je. Ten lze formulovat až v podtřídách, přičemž každý objekt interpretuje smysl takové virtuání metody podle toho, do které podtřídy patří.

Jak už jsme výše naznačili, je OOP lidskou programovací činností. Pod pojem OOP tedy zahrnujeme jak tvorbu tříd, tak jejich aplikaci, a ta může být zaměřena jak na tvorbu dalších tříd, tak na použití tříd k tvorbě konkrétních programů. Těmito programy mohou být i počítačové modely, mimo jiné i modely simulační. Soubor vhodně skloubených tříd může být definicí simulačního jazyka. Jak už jsme uvedli, od každé třídy lze v OOP vytvořit libovolný počet instancí. (Je-li nějaký programovací prostředek omezen tak, že od některých jeho tříd lze vytvořit jen pevný počet instancí, nejde o prostředek zaměřený na OOP a jeho použití není OOP.) Použití OOP tedy vede k zavádění jazyků základního typu T, neboť aktivity v modelech sestavených s pomocí OOP neexistují, vždy jde o transakce degenerované tak, jak bylo naznačeno v závěru oddílu 7.2.5.

7.3.2 Dnešní situace

Dnes jsou v oblibě prostředky pro OOP amerického původu, jako je SmallTalk, C++, JAVA nebo novější verze Pascalu. Tyto prostředky vycházejí z více či méně radikálního pojetí OOP, dle něhož jediná možnost dění v počítači spočívá ve vysílání zpráv a v reakcích adresátů na ně. Kvaziparalelní systémy se v těchto jazycích nepřipouštějí. Při tomto přístupu se tvorba simulačních prostředků omezuje na prostředky s elementárními transakcemi (viz odstavec 7.2.6). Jinými slovy, při použití prostředků pro OOP, které jsou dnes rozšířeny, se sestavují konfigurace tříd, pomocí nichž jsou realizovány různé jazyky, v nichž existují jednak pasivní transakce (transakce odpovídající prvkům, které „pozorujeme“ na simulovaném systému, ale bez vlastních životních pravidel), jednak elementární transakce, které nesou útržky děje v systému.

Pasivní transakce jsou instance tříd, zatímco metody, které tyto transakce provádějí (na podněty odjinud), odpovídají elementárním transakcím.

Můžeme říci, že jazyky takto implementované jsou jazyky typu TT_E. A na rozdíl od nich můžeme též jazyky, o nichž jsme se zmínili v odstavci 7.2.5, prohlásit za jazyky typu T. Jazyky typu TT_E spolu s jazyky typu T jsou jedinými dvěma podskupinami jazyků základního typu T.

7.3.3 Kvaziparalelní programování v jazyku SIMULA

V šedesátých letech se tak rozrůstaly požadavky na tvorbu simulačních programů, že jim pracovní kapacity nestačily, a to ani v oblasti vlastního sestavování simulačních programů, ani v oblasti implementace vhodných simulačních jazyků. A tak v roce 1967 vznikl v Norsku jazyk SIMULA 67 [45, 10, 23, 24, 25, 7], který měl nejen všechny vlastnosti OOP, jak jsme je formulovali v oddíle 7.3.1, ale i prostředky kvaziparalelního programování. Jméno SIMULA dostal proto, že jeho ryze simulační prostředky byly takřka přesně převzaty ze simulačního jazyka SIMULA, o němž jsme se zmínili v odstavci 7.2.5, a přívlastek 67 dostal proto, aby se od uvedeného simulačního jazyka (který byl realizován už v roce 1964) odlišil i svým jménem. Od poloviny osmdesátých let se přívlastek 67 přestal používat, neboť všichni, kdo před tím používali simulační jazyk SIMULA, přešli na použití jazyka SIMULA 67 a simulační jazyk SIMULA se stal historickou položkou.

Jazyk SIMULA 67, který tedy budeme také nazývat jen SIMULA, obsahuje prostředky, které ho kvalifikují jako jazyk základního typu T. Jeho objektově orientovaných nástrojů bylo ovšem už vícekrát využito i k definici dalších simulačních prostředků, pomocí nichž se dá SIMULA aplikovat jako jazyk typu AT nebo typu TA (dá se ovšem aplikovat i jako jazyk typu TT_E, což je ovšem z komerčního hlediska bezvýznamné).

Ten, kdo nezná jazyk SIMULA, může být zvědav, jak se vlastně spojení kvaziparalelního programování s OOP realizuje. Odpověď, byť poněkud zjednodušená, je prostá. Každá třída obsahuje jistou zvláštní, jakoby virtuální metodu, která se neaktivuje pomocí vyslání zprávy, ale vždy tehdy, když je generována instance této třídy. Taková instance ji provádí jako jinou metodu, což je první přiblížení k tomu, že tato metoda představuje vlastně životní pravidla. Provádění takových metod různými prvky je organizováno v rámci kvaziparalelního systému, takže v nich mohou být aplikovány plánovací příkazy. SIMULA připouští i specializaci – čili obohacování a doplňování – takto pojatých životních pravidel v podtřídách.

Spojení prostředků pro OOP s kvaziparalelním programováním je v dnešní době výjimkou. Kromě jazyka SIMULA nabízí ještě jakési zárodky jazyk BETA dánsko-norského původu, a německo-novozélandská nástavba C-FLAVOURS

nad jazykem FLAVOURS. Simulační prostředky těchto jazyků však nejsou zdaleka tak vypracovány jako u jazyka SIMULA.

Vynález OOP byl stimulován potížemi a stále se zvyšujícími požadavky, které na programování kladla simulace a které při tom potvrdily, jak vhodné pro simulaci je kvaziparalelní programování. Celosvětové přijetí OOP však nastalo téměř dvacet let po tom, co simulace podnítila vznik prvního objektově orientovaného jazyka SIMULA, přičemž kvaziparalelní programování bylo od této doby až do dneška v populárních přístupech k OOP ignorováno. Tento dějinný paradox vypovídá o tom, že programování simulačních modelů se dosti liší od jiných programovacích přístupů a technik. Lze tedy očekávat, že překlenutí jisté propasti mezi tvorbou simulačních programů a ostatními směry programování bude v budoucnosti zdrojem mnoha překvapení a podnětů ve vývoji programování vůbec.

7.4 Řízení simulační studie

V předcházejících oddílech této kapitoly jsme se zabývali způsoby popisu simulovaného systému, který je v počítačovém systému přeložen a prováděn jako simulační pokus. Tento termín byl zaveden v odstavci 1.6 a tamtéž jsme zavedli i termín simulační studie pro cílenou posloupnost simulačních pokusů. Je evidentní, že popisu simulovaného systému může – a to v opakovaném režimu – odpovídat i simulační studie, avšak vzniká otázka, zda a jaké jsou prostředky, které usnadňují její programování, tj. které jsou zaměřeny na to, že simulační studie je cílená, že neopakuje simulační pokusy neuspořádaně, nýbrž kvůli jistému záměru. Je evidentní, že – podobně jako simulované systémy – mohou i cíle simulační studie pokrývat široké spektrum možností.

Než postoupíme ve výkladu dále, uvedeme příklad, který pak zobecníme. Předpokládejme, že chceme najít nějakou optimální strukturu továrny, která má být postavena. Simulační studie tedy bude složena ze simulačních pokusů, z nichž každý bude odpovídat jedné variantě továrny. Není ani vyloučeno, že simulační studie bude organizována tak, že se z dosavadních simulačních pokusů bude „učit“ a bude postupně navrhovat další varianty továrny, které by měly být stále lepší a lepší (což je ovšem vždy otázka, na níž dá odpověď právě simulační pokus s danou variantou). Když popisujeme simulační pokus, popisujeme v simulačním jazyku továrnu. Kdybychom však chtěli podobným způsobem popsat simulační studii, museli bychom popisovat situaci, v níž jako by byla realizována jedna továrna (tj. první varianta, odpovídající prvnímu simulačnímu pokusu), s ní by se měly udělat nějaké zkušenosti a pak by se měla likvidovat, na základě zmíněných zkušeností by se měla realizovat jiná, snad lepší továrna (druhá varianta, odpovídající druhému simulačnímu pokusu), po

získání zkušeností s ní by se měla na jejím místě realizovat další továrna a tak dále, dokud se nepostaví továrna vskutku perfektní. Podobně by se měla popsat např. simulační studie týkající se nádorové terapie (pacient je léčen jedním způsobem, pak likvidován, znovu oživen ve stavu před léčbou a na základě zkušeností s první léčbou léčen jinak atd.).

Ačkoliv na počítači lze nasimulovat ledacos, i výše uvedené drastické procesy, je evidentní, že uživatele simulačních programovacích prostředků, kteří nejsou odborníky v informatice a mají představy, které se shodují s realitou jejich profese, by nutnost představ takových drastických procesů spíše odradila a znechutila, než aby jim pomohla. Celá záležitost je navíc komplikovaná ještě tím, že simulační pokus začíná ve všech simulačních prostředcích časem rovným nule a na tomto předpokladu jsou založeny takřka všechny pomocné výpočty zabudované do simulačních jazyků (hodnocení a aktualizace histogramů, výpočty průměrů hodnot, které se mění v čase atd.). Takže při likvidaci jedné varianty továrny, pacienta či jiného simulovaného objektu a jeho následného vytvoření s jinými vlastnostmi by bylo třeba, aby uživatel zlikvidoval a znovu vytvořil zmíněné histogramy a další pomocné entity a také posunul časovou osu tak, aby další varianta simulovaného systému začala existovat v čase nula a nikoliv v čase, kdy byla likvidována varianta předcházející.

Je tedy zřejmé, že právě popsaný přístup by uživateli mnoho nepomohl, a tak je používán jen zřídka, např. v simulačních prostředcích vybudovaných na bázi populárních jazyků pro OOP, o nichž jsme se zmínili v odstavci 7.3.2.

Shrňme obecné poznatky z předcházejícího příkladu. Simulovaný systém popisujeme jako něco, o čem předpokládáme, že by to mohlo reálně existovat, dokonce v newtonovském čase. Když chceme takový popis rozšířit na simulační studii, dostáváme se kamsi mimo fyzikální svět, v němž simulovaný systém může existovat; dostáváme se do světa, kde neplyne čas tak, jak jsme zvyklí a jak to aplikujeme i ve všech oborech vědy, techniky i řízení společnosti s výjimkou několika oblastí moderní fyziky. V takovém světě už např. druhý simulační pokus studie předpokládá, že popsaný systém z prvního pokusu nenávratně zmizí, čas se vrátí, vznikne jiný systém – možná ovšem s vlastnostmi, které získal na základě „poučení“ z dějů, které nastaly v prvním systému – a ten se začne nějak v čase měnit. Návrat času (resp. zcela nový čas) i poučení z „minulé budoucnosti“ (např. ze stavu systému z prvního pokusu, který nastal v okamžiku, do kterého druhý pokus dosud nedospěl) drasticky nesouhlasí s vlastnostmi fyzického světa, které jsou zahrnuty do sémantiky simulačního jazyka a které obsahují vztahy, na nichž jsou založeny tak běžné faktory myšlení jako kauzalita a běh newtonovsky chápaného času. Aby dostali uživatelé simulačních prostředků možnost popsat řízení simulační studie tak „hmatatelně“, jako mohou popisovat simulační pokus, museli by své „vidění světa“ rozšířit kamsi mimo ten fyzikální svět, v němž simulovaný systém může existo-

vat, museli by být vedeni např. k představám, že je nějaká metafyzicky vysoko postavená bytost, která si pohrává s různými fyzickými světy, z nichž některé mohou případně i existovat paralelně vedle sebe, ruší tyto světy (včetně jejich času a objektů, které jsou v nich) a nahrazuje jinými a přitom je porovnává, dokud nevybere ten nejlepší.

Přestože něco podobného cítil ve svých filozofických představách již v 17. století jeden ze zakladatelů infinitesimálního počtu G. W. Leibniz, tvůrci simulačních jazyků se takovýchto představ obávali, a tak pro řízení simulační studie nabízeli více nebo méně jednoduché algoritmické prostředky. Z nich nejjednodušší vedou uživatele k tomu, že sestaví jakousi tabulku parametrů a simulační studie postupuje tak, že provede první pokus s parametry z první řádky tabulky, pak druhý pokus s parametry z druhé řádky tabulky a tak pokračuje, dokud tabulku nevyčerpá. Výsledky jednotlivých pokusů řadí do histogramů a počítá z nich průměry a maxima. Nejsložitější prostředky vedou naopak uživatele k tomu, že chápe simulační studii jako algoritmizovaný cyklus, v němž vystupuje simulační pokus jako jedna složka, a to dle následujícího schématu:

1. příprava parametrů prvního pokusu;
začátek cyklu: 2. provedení pokusu;
3. zhodnocení výsledků pokusu;
4. rozhodnutí, zda ve studii pokračovat; pokud ano:
5. příprava parametrů pro další pokus;
6. skok na začátek cyklu.

Kroky 3 až 5 mohou být velmi složité a mohou používat globální proměnné, na které se mimo jiné mohou ukládat výsledky provedených pokusů i parametry, které další pokusy ovlivní.

Avšak představu oné bytosti, jež by byla nadřazena různým světům, které navzájem fyzicky nesouvisejí, ale jsou v mysli této bytosti porovnávány, tvořeny a likvidovány, je přece jen možno realizovat, a to v objektově orientovaném jazyku SIMULA. V něm totiž existují dva druhy hierarchií, které v jiných simulačních ani objektově orientovaných jazycích neexistují.

- Hierarchie do sebe vnořených tříd: třída může kromě atributů, metod a životních pravidel obsahovat také definice jiných tříd. Taková třída se nazývá **hlavní třídou** a její instance představují **inteligentní činitele** (angl. **intelligent agents**), jež „myslí“ a přitom používají pojmů zobrazených třídami obsaženými uvnitř této hlavní třídy.

- Hierarchie kvaziparalelních systémů: jeden kvaziparalelní systém může obsahovat prvky, které samy obsahují kvaziparalelní systémy.

Jelikož prostředky prvního druhu umožňují modelovat systémy složené z prvků, jež „myslí“, umožňují modelovat i systémy složené z inteligentních činitelů, které si představují nějaké složité děje. Kombinace těchto prostředků s prostředky druhého druhu umožňuje modelovat systémy, jejichž inteligentní činitelé si představují děje, jejichž složitost vzniká paralelním vzájemným ovlivňováním v čase. Jinými slovy, takoví inteligentní činitelé jsou modelováni, jako by simulovali. V jazyku SIMULA lze tedy simulovat (či jiným způsobem modelovat) systémy složené z prvků, z nichž některé či všechny mají své „soukromé“ simulační modely, vzájemně na sobě nezávislé, a manipulují s nimi.

S pomocí jazyka SIMULA lze tedy simulační studii chápat, popsat a na počítači realizovat jako dynamický systém, jehož prvky jsou jednotlivé simulační pokusy. Prostředky pro OOP lze aplikovat nejen na úroveň simulačních pokusů, ale i na úroveň simulační studie, jejíž složitost může tedy nabývat extrémní výše.

V anglosaské literatuře, která pojednává o obecných prostředcích pro simulaci, se můžeme často setkat s termínem **experimental frame**. (Český ekvivalent **experimentální rámec** se u nás příliš neujal.) Tento termín vyjadřuje obecně činnost vztaženou k automatizaci experimentování se simulačním modelem. Často pod tímto termínem myslí autoři právě řízení simulační studie, avšak naplatí to obecně. Slova experimental frame jsou totiž libovolně interpretována v plném významu, co znamenají v angličtině, takže někdy je autoři chápou i např. jako formulaci výstupu výsledků během simulačních pokusů. V osmdesátých létech totiž vznikly tendence oddělit popis vlastního simulovaného systému od popisu experimentování s jeho simulačním modelem a pro každý z popisů použít jiného programovacího jazyka. Popis simulovaného systému sice odpovídá – jak jsme zdůraznili v odstavci 7.2.1 – popisu simulačního pokusu, ale během takových pokusů se může vyskytnout výstup výsledků, aktualizace histogramů, sběr a integrování některých parametrů v simulovaném čase apod. a popis těchto akcí měl být oddělen od popisu vlastního systému a připojen k popisu řízení simulační studie.

Uveďme příklad. Při simulaci továrny popisujeme simulační pokus tak, že popíšeme onu továrnu včetně toho, jako bychom během její existence přímo v ní dělali nějaké pokusy, měření, detekce atd. Byly tendence oddělit vlastní popis továrny od popisu těch pokusů, měření a detekcí prováděných v ní a tyto akce připojit k popisu simulační studie, tj. k popisu strategie, jak např. na základě údajů o chování jedné varianty továrny vytvořit variantu jinou, snad lepší. Z jistého pohledu na věc byla idea oddělení popisu experimentů od popisu továrny vhodná: stává se totiž, že na základě nějakých výsledků pokusů s továrnou nás napadne to, že bychom s ní ještě měli udělat jiné pokusy, než ji změníme, než přejdeme na jinou variantu. Kdyby byl popis experimentování oddělen od popisu továrny, změnili bychom v takovém případě pouze ten popis

experimentování.

Avšak chtít se vyjadřovat podobným jazykem jak o experimentování se systémem během jeho existence, tak o vytváření dalších variant systému je velmi násilný přístup. Nadto virtuální procedury (čili – jak se v některých prostředcích pro objektově orientované programování říká – metody s dynamickými či pozdními vazbami) vyřešily problém výměny experimentálních kroků v popisu simulovaného systému. Možná místa experimentů v systému se prohlásí za virtuální procedury a jejich náplň se pak může libovolně měnit, a to odděleně od popisu simulovaného systému (a bez zásahů do něho). A tak experimentální rámec jako cosi svébytného a s vlastním jazykem se udržel jen u některých simulačních jazyků, z nichž nejznámější v USA je DYMOLA.

7.5 Poznámka ke kombinované simulaci

V této kapitole jsme mezi jednotlivými druhy simulačních jazyků uváděli pouze jazyky pro spojitou simulaci (viz odstavce 7.2.3.1 a 7.2.3.2) a jazyky pro diskrétní simulaci (všechny ostatní). Nezmínili jsme se o jazycích pro kombinovanou simulaci, abychom klasifikaci z hlediska vztahu k aktivitám a transakcím příliš nezatemnili.

V kapitole 1 jsme naznačili, že kombinovaná simulace se chápe jako simulace, kde simulovaný systém má podstatné vlastnosti jak systémů spojitých, tak i systémů diskrétních. Kombinovaná simulace není tedy jen nějaké malé „poškození“ spojité simulace nějakými nespojitostmi nebo nějaký „snad“ spojitý fenomén v jinak diskrétním systému (např. rovnoměrný či rovnoměrně zrychlený pohyb transakcí, při jehož modelování by stačilo počítat prostoročasové souřadnice výchozích a výsledných - tedy diskrétních - stavů řešením rovnic, které se každý může naučit v prvních týdnech středoškolských hodin fyziky).

Z těchto požadavků, jež klade na kombinovanou simulaci světová odborná veřejnost, vyplývají i požadavky na odpovídající programovací prostředky: jazyky pro kombinovanou simulaci musí povolit jednak popis systémů s transakcemi, jednak popis spojitých změn určených nejméně obyčejnými diferenciálními rovnicemi. Z prvního požadavku plyne, že jazyky pro spojitou simulaci jsou základního typu AT nebo základního typu T. Druhá vlastnost souvisí s otázkou, která existuje už v souvislosti se spojitou simulací, jak široce se vlastně má pojem spojitosti v simulaci chápat. Odpověď na tuto otázku souvisí s odpovědí, kterou dává právě skutečnost prostředků pro spojitou simulaci, kde se spojitost chápe jako chování podle obyčejných diferenciálních rovnic s derivacemi podle času. I v kombinované simulaci jsou takové rovnice postačujícím důvodem pro přijetí spojitosti. Při psaní těchto učebních textů byl znám jediný

jazyk pro kombinovanou simulaci (GASP V), který připouští, aby jeho uživatel popisoval i parciální diferenciální rovnice, zatímco všechny ostatní jazyky pro kombinovanou simulaci jsou omezeny jen na diferenciální rovnice obyčejné.

Řečeno prozatím velmi stručně, v kombinovaném systému tedy existují prvky, jejichž některé atributy se mohou měnit (alespoň někdy) spojitě a dále v něm existují diskrétní události, včetně vzniku a zániku transakcí. Avšak to není všechno. Vyskytují-li se v nějakém systému spojité a diskrétní změny, lze právem očekávat, že se v něm tyto změny dostanou i do nějaké vzájemné závislosti, a tak jazyky pro kombinovanou simulaci musí být vybaveny prostředky pro popis těchto závislostí. Jde konkrétně o dva druhy závislostí.

- Diskrétní událost způsobí diskrétní změnu hodnoty atributu, která se jinak mění spojitě.
- Spojitě se měnící atribut dosáhne jistého „kritického“ stavu, což způsobí diskrétní událost, která se může projevit na zcela jiných vlastnostech systému než na daném atributu. Taková událost se nazývá **stavovou událostí** (anglicky **state event**). Nevylučuje se, že kritická událost může aktivovat čekající proces nebo naopak pasivovat jiný proces včetně toho, jehož atribut k této události svým spojitým průběhem vedl. Nevylučuje se ani to, že kritická událost způsobí vznik nových transakcí nebo naopak likvidaci jiných včetně té, jejíž atribut tuto událost způsobil.

Pro diskrétní změnu hodnoty atributu, který se jinak mění spojitě (případ 1) není třeba vymýšlet nějaký nový vyjadřovací prostředek. Jde o zcela obvyklé dosazení, stejně vyjádřené jako dosazení nové hodnoty za atribut, který se spojitě nemění. V implementaci jazyka to ovšem může znamenat jisté komplikace, ale o ty se uživatel nemusí starat (a proto si jich v této kapitole nevšímáme).

Pro stavovou událost se nabízí několik možností. Jazyky s elementárními prvky (viz odstavec 7.2.6) mohou snadno nabídnout uživateli, aby stavovou událost popsal jako elementární transakci, pro kterou platí jistá podmínka, kdy nastane, a to podmínka, že daný atribut dosáhne kritické hodnoty. Ostatní simulační jazyky (tedy konkrétně jazyky základního typu AT či T) řeší celou věc tak, že jeden nebo více prvků, které provádějí svá životní pravidla, narazí na jistý speciální druh interrogativního plánovacího příkazu (viz odstavec 7.2.4), totiž příkazu „čekej, až daný atribut dosáhne jisté (totiž té kritické) hodnoty“.

Když se nad stavovými událostmi hlouběji zamyslíme, dojdeme k názoru, že stavová událost by mohla nastat obecně v okamžiku, v němž hodnoty nějakých atributů vyhoví nějaké podmínce, čili když splní nějaký booleovský výraz. Takový výraz přece nemusí být jen ta nejjednodušší relace, tedy relace tvaru $a = k$, kde a je uvažovaný atribut a k je konstanta, nýbrž např. $a1 = a2$ (kritická událost nastane, když se hodnoty dvou atributů budou sobě rovnat),

nebo $a1 = k1 * a2 + k2$, kde $a1$ a $a2$ jsou nějaké atributy (dokonce nemusí být oba současně spojité) a $k1$ a $k2$ nějaké konstanty (nebo dokonce také atributy).

Na dotazy, které při takovém rozboru v mysli každého mohou vzniknout, odpovídají jednoznačně „historická“ fakta o jazycích pro kombinovanou simulaci. Zdá se, že se všechny dnes existující jazyky pro kombinovanou simulaci omezují právě jen na ten jednoduchý případ dosažení kritické hodnoty, případně povolují zadat i směr, odkud je tato hodnota dosažena (událost nastane, je-li hodnota dosažena zdola, resp. shora, resp. na směru dosažení nezáleží). Možnost složitější formulace podmínky pro vznik stavové události se dnes uživatelům nenabízejí a ti si musí s takovými případy poradit sami (v praxi ovšem jsou podobné případy spíše výjimečné).

Na závěr uveďme klasifikaci jazyků pro kombinovanou simulaci, která vyjadřuje jakousi schopnost jazyka popsat více nebo méně. Budeme mluvit o třech kombinovaných typech jazyka, a to K1, K2 a K3.

Jazyky kombinovaného typu K1 povolují popsat systémy složené z jedné „spojité“ aktivity C a z „diskrétního podsystému“ D. C má atributy, jejichž změny se mohou popsat pomocí diferenciálních rovnic, kdežto v D nic podobného neexistuje, to je vskutku diskrétní systém. Prvky v D však mohou způsobit diskrétní změny atributů v C a atributy v C mohou vést ke stavové události v D. Ve Spojených státech populární jazyky GASP IV a jejich odvozeniny jsou tohoto typu, stejně jako výše zmíněný GASP V.

Jazyky kombinovaného typu K2 povolují, aby jakákoliv transakce měla atributy, které se mění spojitě podle diferenciálních rovnic vztažených k této transakci (v praxi: k třídě, jejíž je transakce instancí). Mohli bychom říci, že jazyky typu K2 se liší od jazyků typu K1 tím, že prvek nesoucí spojitě se měnící atributy nemusí být jeden a počet takovýchto prvků se může měnit - mohou vznikat a zanikat. Závislosti obou výše popsaných typů mohou ovšem existovat mezi transakcemi. Transakce bez spojitě se měnících atributů je vlastně degenerovaným případem transakcí se spojitě se měnícími atributy. Mezi tyto jazyky patřil první vůbec v historii navržený jazyk pro kombinovanou simulaci (z roku 1968 a asi nikdy neimplementovaný) a několik jazyků definovaných pomocí podtříd třídy SIMULATION v objektově orientovaném jazyku SIMULA (viz odstavec 7.3.3).

Poznámka. Než si všimneme jazyků kombinovaného typu K3, musíme si uvědomit, že spojité vztahy mezi atributy (vyjádřené pomocí systémů diferenciálních rovnic) nastávají jen uvnitř aktivity D (v případě typu K1), resp. uvnitř jednotlivých transakcí (v případě jazyků typu K2). Vztahy mezi transakcemi navzájem a k ostatním prvkům se projevují vždy jen jako diskrétní události.

Toto pravidlo porušují jazyky typu K3: ty připouštějí, aby mezi sebou spojitě reagovaly atributy různých prvků, což je převratné právě u transakcí. Nechť

a je atribut transakce P a b je atribut jiné transakce Q a oba jsou spojeny nějakým systémem diferenciálních rovnic (jež se ovšem obecně může vztahovat i k dalším atributům spojitě se měnícím podle rovnic tohoto systému). Když transakce Q ze systému zmizí, zmizí i atribut b a uvedený systém rovnic přestane mít smysl. Musí být nahrazen jiným systémem. Něco podobného by mělo nastat i v tehdy, když transakce Q do simulovaného systému vstoupila a „zaangažovala svůj atribut b“ do nějakého systému diferenciálních rovnic: ten ovšem musel před tím mít jiný tvar, nemohl obsahovat b.

U typu K3 se tedy setkáváme se spojitými změnami podle diferenciálních rovnic, jejichž systém se v čase mění. Prostředky pro popis něčeho podobného zatím v naší civilizaci neexistují, avšak systémy, v nichž diskrétní změny způsobují kvalitativní změny pravidel pro spojité změny, kolem nás existují, a to velmi často - příkladem z průmyslu může být hlubinná pec v ocelárnách, v níž jsou zahřívány bramy, jejich počet - a tudíž i vzájemné přenášení tepla - se v čase mění. Z toho důvodu jsou jazyky typu K3 velmi aktuální. Proměnná pravidla pro spojité změny se v nich úspěšně obcházejí tak, že uživatel jazyka popisuje transakce jako zapojení prvků ideálního analogového počítače (sčítaček, integrátorů, násobiček, invertorů, generátorů funkcí atd.), které mohou v čase měnit svou strukturu (mohou i libovolně narůstat, pulzovat co do své velikosti apod.). Tyto jazyky jsou vybaveny i možností definovat makra (vybraná zapojení) a dynamicky je zapojovat všude, kde je možno zapojovat původní, „elementární“ prvky idealizovaného analogového počítače. Prvním jazykem tohoto typu byl NEDIS ukrajinské provenience, který se v něčem nechal inspirovat třídami v jazyku SIMULA (ale ne podtřídami a lokálními třídami), po něm následovalo několik jazyků budovaných přímo na bázi jazyka SIMULA (německý SIMKOM/S, portugalský CDS, norský COMBINEDSIMULATION a český BOSCOS).

Jazyky typu K3 se dnes začínají používat také v aplikaci tzv. přímkové metody pro simulaci spojitých systémů popsatelných pomocí parciálních diferenciálních rovnic. Přímková metoda spočívá v diskretizaci takového systému, při níž se derivace podle jiných hodnot než času aproximují diferencemi, takže z parciální rovnice vznikne soustava rovnic obyčejných. Vzhledem k tomu, že odstraněné parciální derivace mohou měnit své hodnoty v čase, je nutno měnit i počet aproximujících obyčejných rovnic, jež však spolu interagují spojitě, takže možnosti, které nabízejí jazyky typu K3, nelze v tomto případě obcházet pomocí nástrojů, jež nabízejí jazyky typu K1 a K2.

Literatura

[1] Anděl, J. *Statistická analýza časových řad*. Praha: SNTL, 1976. 272 s.

[2] Arnoľd, V. I. *Dopoľniteĺnyje glavy teorii obyknovennych differenciaľnych uravnenij*. Moskva: Nauka, 1978.

[3] Ashby, W. R. An Introduction to Cybernetics. New York: Wiley & Sons, 1963.

[4] Bailey, N. T. J. *The Mathematical Theory of Infectious Diseases*. New York: Hafner Press, 1975.

[5] Brunovský, P. Topologická klasifikácija diferenciálných rovnic a štruktúrálná stabilita. *PMFA*, 1973, roč. 18, s. 271.

[6] Brunovský, P. Bifurkácije negradientných dynamických systémov. *PMFA*, 1982, roč. 27, s. 74.

[7] Cendelín, J., Kindler, E. *Modelování a simulace*. Skripta ZČU Plzeň. Plzeň: Ediční středisko ZČU, 1994. 230 s.

[8] Cipra, T. *Analýza časových řad s aplikacemi v ekonomii*. Praha: SNTL, 1986. 248 s.

[9] Dahl, O.-J., Nygaard, K. *SIMULA, a language for programming and description of discrete event systems*. 5. vydání. Oslo: Norsk Regnesentralen, 1967.

[10] Dahl, O.-J., Myhrhaug, B. Nygaard, K. *Common Base Language*. 1. vydání. Oslo: Norsk Regnesentralen, 1968.

[11] Kolektiv autorů. Dohoda o chápání pojmu simulace systémů. *Automatizace*, 1986, roč. 29, č. 12, s. 299–300.

[12] Donaghey, C. E. *CELLSIM User Manual*. Houston: University of Houston, 1973.

[13] Frauenthal, J. C. *Mathematical Modelling in Epidemiology.* Berlin: Springer-Verlag, 1980.

[14] Gaylord, R. J., Wellin, P. R. *Computer Simulations with Mathematica. Explorations in Complex Physical and Biological Systems.* New York: Springer-Verlag, 1994. ISBN 0-387-94274-2.

[15] Gaylord, R. J., Nishikate, K. *Modeling Nature. Cellular Automata Simulations with Mathematica.* New York: Springer-Verlag, 1996. ISBN 0-387-94620-9.

[16] Hájek, P., Havránek, T. *Mechnizing Hypothesis Formation – Mathematical Foundations of a General Theory.* Berlin: Springer Verlag, 1978.

[17] Hannan, E. J. *Multiple Time Series.* New York: Wiley & Sons, 1971.

[18] Herman, J., Rozenberg, J. *Developmental Systems and Languages.* Amsterdam: North Holland, 1975.

[19] Chytil, M. *Automaty a gramatiky.* Matematický seminář č. 19. Praha: SNTL, 1984.

[20] Johnson, S. C., Wichern, J. *Applied Multivariate Statistical Analysis.* New Jersey: Prentice-Hall, 1982.

[21] Kindler, E. Simulation System COSMO – Description of Its Language and Compiler. *Kybernetika*, 1969, roč. 5, s. 287.

[22] Kindler, E. *Simulační programovací jazyky.* Praha: SNTL, 1980. 280 s.

[23] Kindler, E., Brejcha, M. *Programování a algoritmizace v jazyce SIMULA.* 2 díly. Plzeň: Dům techniky, 1987. 232 s.

[24] Kindler, E., Brejcha, M. *Programování v jazyce SIMULA.* 2 díly. Rozšířené vydání [7]. Plzeň: Ediční středisko ZČU, 1992. 250 s.

[25] Kindler, E., Brejcha, M. *Objektově orientované programování.* Upravené vydání [24]. Plzeň: Ediční středisko ZČU, 1994. 250 s.

[26] Kindler, E. *Soukromé sdělení* (1994).

[27] Kleijnen, J. P. C. *Statistical Techniques in Simulation (in two parts).* New York: Marcel Dekker, 1974.

[28] Kowalski, O. Thomova věta o sedmi elementárních katastrofách. it PMFA, 1977, roč. 22, s. 109.

[29] Knuth, D. E. *The Art of Computer Programming*. Reading: Addison-Wesley, 1969.

[30] Kotva, M. *Obecná metoda multikompartmentových modelů*. Kandidátská disertace. Praha: VŠE, 1979.

[31] Křivý, I. *Obecná metoda multikompartmentových modelů*. Výzkumná zpráva k úkolu ZV I-1-5/04. Ostrava: Pedagogická fakulta, 1986.

[32] Maeder, R. *Programming in Mathematica*. Reading: Addison-Wesley, 1996. ISBN 0-201-85449-X.

[33] Malík, M. **Počítačová simulace**. Skripta MFF UK. Praha: UK Praka, 1989. 535 s. ISBN 80-7066-121-6.

[34] Marek, M., Schreiber, I. *Stochastické chování deterministických systémů*. Praha: Academia, 1984. 160 s.

[35] Mladov, A. G. *Sistemy differenciaĺnych uravnenij i ustojčivosť po Ljapunovu*. Moskva: Vysšaja škola, 1966. 272 s.

[36] Nagy, J. *Stabilita řešení obyčejných diferenciálních rovnic*. Praha: SNTL, 1980. 72 s.

[37] Nekvinda, M., Šrubař, J., Vild, J. *Úvod do numerické matematiky*. Praha: SNTL, 1976. 288 s.

[38] Novák, V. *Fuzzy množiny a jejich aplikace*. Praha: SNTL, 1990. 148 s. ISBN 80-03-0325-3.

[39] Odum, E. P. *Základy ekologie*. Praha: Academia, 1977.

[40] Papert, S. *Mindstorms: Children, computers and powerful ideas*. New York: Basic Books, 1980.

[41] Rábová, Z.et al. *Modelování a simulace*. Skripta FEL VUT Brno. Brno: VUT Brno, 1992.

[42] Ralston, A. *Základy numerické matematiky*. Praha: Academia, 1978. 636 s.

[43] Rosen, R. *Dynamical System Theory in Biology*. Vol. I. *Stability Theory and Its Applications*. New York: Wiley – Interscience, 1970.

[44] Schriber, T. J. *An Introduction to Simulation Using GPSS/H*. New York: Wiley & Sons, 1991.

[45] SIMULA Standard as defined by the SIMULA Standard Group, 25th August 1986. Oslo: Simula a. s., 1989.

[46] Smale, S. Differentiable Dynamical Systems. *Bull. Amer. Math. Soc.*, 1967, vol. 73, p. 747.

[47] Thom, R. *Structural Stability and Morphogenesis*. Reading: Benjamin, 1975.

[48] Thompson, J. M. T. *Neustojčivosti i katastrofy v nauke i technike*. Moskva: Mir, 1985.

[49] Weinberger, J. *Project Management Forecast*. Firemní materiály. Praha: TIMING, 2001.

[50] Weinberger, J. Extremization of Vector Criteria of Simulation Models by Means of Quasi-Parallel Handling. *Computers and Artificial Intelligence*, 1987, vol. 3, no. 1, pp. 71–79.

[51] Weinberger, J. Evolutional Approach to Extremization of Vector Criteria of Simulation Models. *Acta Universitatis Carolinae Medica*, 1988, vol. 34, no. 3/4, pp. 249–258.

[52] Wright, D. J. *Dynamical Systems and Fractals Lecture Notes*. Dostupné na internetu:
$< http://www.math.okstate.edu/mathdept/dynamics/lecnotes/ >$.

[53] Zadeh, L. E. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Trans. Syst. Man. Cybern.*, 1973, vol. 1, p. 28.

[54] Zvára, K. *Regresní analýza*. Praha: Academia, 1989.