Tehnici de modelare şi simulare

Îndrumător de laborator

Rodica Ţirtea
2008

Cuvânt înainte

Tehnicile de modelare şi simulare s-au dezvoltat în ultimele decenii într-un ritm neaşteptat mulţumită creşterii capacităţii de calcul şi a vitezei sistemelor de calcul. În acest fel, utilizând platforme performante pentru modelare şi simulare, costurile de proiectare şi testare sunt reduse în mod substanţial.

Această lucrare propune câteva teme practice pentru a fi realizate la activitatea de laborator a studenţilor care aleg cursul de „Tehnici de modelare şi simulare". Structura materialului urmează capitolele cursului de Tehnici de modelare şi simulare.

Acest îndrumător de laborator abordează diverse teme, de la cele care vizează erorile în modelare, modelarea proceselor independente de timp (utilizând atât automate de stări cât şi reţele Petri), la modelarea sistemelor temporizate şi apoi probabilistice ca în final să se analizeze şi alte domenii în care pot fi utilizate tehnicile de modelare şi simulare. Exemplele şi modelele sunt selectate dintre cele care au aplicaţie şi sunt utile în modelarea proceselor legate de sistemele de calcul.

Prin această lucrare se încearcă familiarizarea cursantului cu noţiunile şi caracteristicile tehnicilor de modelare. Fiecare lucrare propusă recapitulează pe scurt noţiunile prezentate la curs pe acea temă. Temele propuse se bazează, în principal, pe două referinţe bibliografice: *Introduction to Computational Science: Modeling and Simulation for the Sciences*, Angela B. Shiflet & George W. Shiflet, Princeton University Press, 2006 şi respectiv *Introduction to Discrete Event Systems*, Christos G. Cassandras & Stephane Lafortune, Springer, 2008, a doua ediţie. Pentru implementări nu sunt indicate anumite platforme pentru modelare şi simulare pentru a lăsa la latitudinea cursanţilor selectarea diversele instrumente software care pot fi utilizate. În funcţie de domeniul de utilizare a tehnicilor, există diverse platforme profesionale pentru modelare şi simulare. Doar în unele locuri sunt indicate instrumente software, şi în special acestea sunt din domeniul Open Source.

Domeniul de studiu „Tehnici de modelare şi simulare", în mod evident, permite extinderea cursului şi a temelor practice propuse. Totuşi, în realizarea acestui material s-a ţinut cont de planul de învăţământ care specifică patrusprezece ore de laborator, astfel, doar temele considerate esenţiale au fost incluse.

Sper ca acest material să fie pe de o parte util şi pe de altă parte utilizat de către studenţi. Mulţumesc pe această cale studenţilor care până acum au furnizat opinii şi reacţii vis-a-vis de acest material şi pe de altă parte sper ca în continuare să existe feedback din partea celor care îl utilizează.

Rodica Ţirtea
Noiembrie 2008

Cuprins

1. Modelare şi simulare. Scurtă analiză a erorilor în modelare

1.1. Introducere

Pe parcursul soluționării unei probleme de modelare care presupune calcul matematic pot să apără erori. Pot fi influențate de erori toate etapele parcurse în modelare – de la colectarea datelor până inclusiv la implementarea pe calculator. Cei care realizează modelarea sistemelor trebuie să fie conştienți de posibilitatea apariției acestor erori, şi, pe de o parte trebuie să minimizeze influența erorilor iar pe de altă parte să evite formularea unor concluzii eronate pe baza unor rezultate greşite. În această lucrare sunt prezentate şi discutate concepte legate de erori şi surse de erori. Sunt rezolvate exerciții pentru exemplificare. De asemenea sunt propuse, atât pe secțiuni, cât şi la final, exerciții recapitulative.

1.2. Tipuri de erori în modelare[1]

În această secțiune nu ne propunem să realizăm o clasificare a erorilor. Vom analiza totuşi diverse tipuri de erori care apar pe parcursul procesului de modelare şi vom identifica modalitățile cele mai eficiente de a reduce efectul acestor erori.

1.2.1. Date eronate

Pentru a colecta date de intrare, majoritatea sistemele pe care le modelăm utilizează senzori. Valorile furnizate de aceşti senzori sunt apoi analizate şi utilizate pentru decizii. Orice disfuncționalitate a acestor senzori determină decizii eronate. De exemplu, dacă un senzor care detectează presiunea dintr-o instalație nu funcționează corect, sau nu este bine calibrat, nu va putea furniza date corecte pentru dispozitivul de control al presiunii. De asemenea, precizia detecției poate să fie o problemă.

1.2.2. Erori de modelare

Etapa de modelare poate să fie însoțită de erori. Persoanele implicate în procesul de modelare pot să greşească. Dacă modelul este simplificat prea mult atunci ecuațiile care descriu sistemul nu mai corespund procesului inițial şi deci nu mai corespund realității. De exemplu, lordul Kelvin (cel care a introdus sistemul de măsurare a temperaturii cu acelaşi nume) a propus pe la mijlocul secolului XIX un model matematic şi a calculat vârsta planetei noastre ca fiind între 20 şi 40 de milioane de ani. Totuşi aceasta valoare este mult diferită de cea reală de aprox. 12 miliarde de ani. Lordul Kevin a presupus ca Pământul se răceşte (din faza în care era doar o masă incandescentă) şi Soarele este singura sursă de energie. Totuşi această presupunere nu este corectă deoarece descompunerea elementelor radioactive din scoarța terestră generează de asemenea energie termică, încetinind astfel răcirea planetei. Totuşi lordul Kelvin nu a putut lua în considerare aceste efecte ale radioactivității deoarece radioactivitatea a fost descoperită de Becquerel mai târziu, în anul 1896.

[1] *Introduction to Computational Science: Modeling and Simulation for the Sciences*, Angela B. Shiflet & George W. Shiflet, Princeton University Press, 2006 (Module 2.2).

1.2.3. Erori de implementare

Etapa de implementare este de asemenea supusă erorilor. Un exemplu edificator în acest sens este pierderea de către NASA în 1999 a unei navete spaţiale care avea misiunea de a observa planeta Marte. Dezastrul a fost cauzat de faptul că sistemele furnizate de o parte din producători erau gândite utilizând sistemul metric şi unităţi referenţiale iar cealaltă parte a furnizorilor de echipamente a utilizat sistemul englezesc de măsură.

1.2.4. Precizia

Erori pot să apără şi în etapa de calcul datorită preciziei de calcul. Vom analiza mai în detaliu acest aspect.

Multe din limbajele de programare utilizează reprezentarea numerelor în virgulă flotantă în formă exponenţială. De exemplu un rezultat de forma 9.843600e02 însemnând $9.843600 \times 10^2 = 984.36 = 0.98436 \times 10^3$. Numerele cu virgulă flotantă sunt memorate de către calculatoare în trei părţi: prima parte pentru *semn*, 0 sau 1, pentru semnul + sau respectiv -, a doua parte este *mantisa* (sau partea semnificativă sau fracţionară) 98436 în cazul nostru, şi ultima parte este *exponentul*, adică 3 în acest caz.

Notaţia exponenţială se reprezintă ca şi produs a unei parţi zecimale cu o putere de a lui 10 ex. dacă *a* este partea zecimală şi *n* un întreg, notaţia exponenţială *aen* reprezintă $a \times 10^n$. Partea întreagă rezultată prin eliminarea virgulei din *a* formează *mantisa* iar *n* reprezintă *exponentul*.

Forma normalizată este dată de scrierea în care mantisa are virgulă în faţă primei cifre diferite de zero: ex. $0{,}98436 \times 10^3$. În această formă, toate cifrele diferite de zero sunt semnificative ex. 98436. Pentru numere întregi cum ar fi $003\ 704\ 000 = 0.3704 \times 10^7$, cifrele 3704 sunt *cifre semnificative*, Iar 3 este *cea mai semnificativă cifră.* Pentru numerele reale, de exemplu pentru $0{,}09200 = 0{,}92 \times 10^{-1}$, 9 este cea mai semnificativă cifră iar 9200 sunt cifrele semnificative (deci inclusiv 0 la final).

Forma ştiinţifică în schimb plasează virgula după prima cifră diferită de zero, astfel încât aceeaşi valoare este scrisă $9{,}8436 \times 10^2$.

Precizia este dată de numărul de cifre semnificative. Astfel atât 003 704 000 cât şi 0,09200 au aceeaşi precizie, adică 4.

Magnitudinea indică mărimea relativă a valorii şi este 10 la puterea dată de exponent în forma normalizată. Astfel 0.3704×10^7 are magnitudinea 10^7.

În C respectiv C++ precizia unui număr în virgulă flotantă de tipul *float* (adică precizie simplă) este de 6-7 cifre zecimale pentru partea semnificativă în timp ce magnitudinea variază între 10^{-38} şi 10^{38}. Tipul *double* utilizează spaţiu de memorie dublu pentru memorie comparativ cu tipul *float* şi utilizând dubla precizie poate fi utilizat pentru a reprezenta 14-15 cifre semnificative iar magnitudinea poate varia intre 10^{-308} şi 10^{308}.

Întrebări recapitulative 1. Pentru valurile (i) 0,0004500; (ii) 230000; (iii) 0,312 să se rezolve următoarele probleme:

a. Utilizând forma normalizată să se specifice cifrele semnificative.

b. Utilizând forma normalizată să se specifice exponentul lui 10.

c. Să se specifice precizia.

1.2.5. Erori absolute şi erori relative

În cazul în care apar erori este important să existe modalităţi pentru a determina magnitudinea acestora. *Eroarea absolută* este dată de valoarea absolută a diferenţei dintre rezultatul calculat şi cel exact/corect. *Eroarea relativă* se determină prin împărţirea valorii absolute a diferenţei raportat la valoarea exactă (presupunând că nu este zero).Eroarea relativă se reprezintă în general ca şi procent ex. o eroare de 0.04 se reprezintă ca şi 4%.

Deci, dacă valoare corecta o notăm ‚corect' şi cea calculată o notăm ‚calculat' avem:

eroarea absolută = | corect – calculat |

eroarea relativă = (eroarea absoluta) / | corect | = | corect – calculat | / | corect |

sau

eroarea relativă = (eroarea absoluta) / | corect | x 100% = | corect – calculat | / | corect | x 100%

presupunând *corect* $\neq 0$.

Exemplul 1. Presupunem că un calculator are precizia 3 – cea ce înseamnă că permite reprezentarea a doar 3 cifre semnificative şi *trunchează* biţii mai puţin semnificativi dacă sunt mai mulţi decât 3. (Acesta este doar un exemplu şi pentru a simplifica calculul s-a făcut această presupunere, în realitate calculatoarele au precizie mult mai mare).

Trunchierea reprezintă eliminarea biţilor mai puţin semnificativi în afară de cei *n* permişi.

Ca şi exemplu vom calcula eroarea relativă şi absolută pentru înmulţirea $(0{,}356 \times 10^{8})(0{,}228 \times 10^{-3})$. Rezultatul exact este:

$(0{,}356 \times 10^{8})\ (0{,}228 \times 10^{-3}) = (0{,}356)\ (0{,}228)\ (10^{8})\ (10^{-3}) = 0{,}081168 \times 10^{5}$.

În formă normalizată se obţine: $0{,}81168 \times 10^{4}$.

Din cauza limitării de precizie calculatorul va returna rezultatul $0{,}811 \times 10^{4}$.

Astfel eroarea absolută este:

$|\text{corect} - \text{calculat}| = |0{,}81168 \times 10^{4} - 0{,}811 \times 10^{4}| = 0{,}00068 \times 10^{4} = 6.8$

iar eroarea relativă:

$(0{,}00068 \times 10^{4})/(0{,}81168 \times 10^{4}) = 0{,}0008378 = 0{,}08378\%$.

Întrebări recapitulative 2. Pentru valorile (i) 6,239; (ii) 0,312 să se calculeze:

a. Eroarea absolută în cazul în care se trunchează la 2 cifre.

b. Eroarea relativă (exprimaţi rezultatele în procente).

1.2.6. Erori round-off

În unele cazuri calculatoarele pot să *rotunjească* valorile în loc să le trucheze pentru a corespunde unei precizii predefinite. Pentru ca valoarea $0{,}81168 \times 10^{4}$ să fie rotunjită cu o precizie de 3 cifre se analizează valoarea celei de a patra valori – în cazul acesta 6. Dacă cifra e mai mică decât 5 se *rotunjeşte în jos* iar dacă este mai mare sau egală se *rotunjeşte în sus*. Astfel $0{,}811\mathbf{6}8 \times 10^{4}$ respectiv $0{,}811\mathbf{5}8 \times 10^{4}$ rotunjite devin $0{,}81\mathbf{2} \times 10^{4}$ iar $0{,}811\mathbf{3}8 \times 10^{4}$ devine $0{,}81\mathbf{1} \times 10^{4}$.

Întrebări recapitulative 3. Rotunjiți valorile care urmează ştiind că valoarea preciziei este 2:

a. $0{,}93742 \times 10^{-5}$ b. $0{,}93472 \times 10^{-5}$ c. $0{,}93572 \times 10^{-5}$

Exemplul 2. În urma unei instrucțiuni de atribuire cum ar fi x= 1.0 / 3.0 (în programe ca şi C, C++, Java) calculatorul va memora la locația lui x valoarea reală 1/3=0,333… . Acelaşi lucru se întâmplă pentru un program cum este Excel unde dacă într-o celulă s-a introdus =1/3 se va memora rezultatul împărțirii. Dacă calculatorul poate memora doar cu o precizie de 3 cifre, calculatorul va rotunji sau trunchia rezultatul la valoarea 0,333.

Eroarea round-off este dată tocmai de acest fapt – calculatorul nu are alocați suficienți biți pentru a memora numărul real în întregime şi atunci utilizează cel mai apropiat număr care poate fi reprezentat.

1.2.7. Erori overflow şi underflow

Aceste erori sunt de asemenea determinate de spațiul limitat de memorare cu care se lucrează. De exemplu dacă se lucrează cu un calculator foarte mic care utilizează doar 16 biți pentru a reprezenta un întreg, dacă dorim să adunăm 20480 + 16384 în mod surprinzător rezultatul va fi un număr negativ, adică -28672. Problema apare deoarece cel mai semnificativ bit devine 1 dat fiind transportul de pe nivelul precedent. Dar cum pentru numerele întregi bitul cel mai semnificativ este bit de semn rezultatul este explicabil. Eroarea *overflow* apare de asemenea în cazul în care se adună două numere negative mari şi se obține un rezultat pozitiv.

O eroare de tip overflow a cauzat pierderea rachetei Ariane 5 a Agenției Spațiale Europene în 1996. La mai puțin de 37 de secunde de la lansare sistemul de ghidare a încercat să transforme viteza circulară dintr-o valoare float pe 64 de biți într-un întreg de 16 biți. Deoarece numărul a fost prea mare a rezultat o depăşire a domeniului – o eroare overflow. Sistemul de ghidaj a încercat să corecteze poziția (dintr-una presupus ca fiind greşită – care oricum nu fusese atinsă) şi foarte curând racheta a trebuit să se autodistrugă. O eroare overflow dată de doar câțiva biți a cauzat pierderea unei rachete în care s-au investit zece ani de cercetare şi şapte miliarde de dolari pentru dezvoltare.

Erorile *underflow* apar când o valoare este prea mică pentru a fi reprezentată corect. De exemplu dacă un calculator permite reprezentarea unor valori de magnitudine 10^{-39} şi rezultatul unei operații este de magnitudine 10^{-48} din cauza acestei erori rezultatul returnat va fi 0.

1.2.8. Erori aritmetice

Erori pot să apară la adunarea numerelor. Astfel dacă se realizează adunarea, spre deosebire de înmulțire, trebuie să se realizeze alinierea virgulei zecimale. Spre exemplu, pentru o precizie de 3 cifre, dacă se adună $(0{,}684 \times 10^{3}) + (0{,}950 \times 10^{-2})$ avem:

$$0{,}684 \times 10^{3} = 684{,}0000$$
$$0{,}950 \times 10^{-2} = \underline{\quad +{,}0095}$$
$$684{,}0095$$

La normalizarea rezultatului se obține $0{,}684 \times 10^{3}$, astfel se pierde influenta celui de-al doilea termen $0{,}950 \times 10^{-2}$.

Întrebări recapitulative 4. Presupunem că un calculator rotunjeşte numerele în virgulă flotantă la 4 cifre semnificative. Calculaţi expresia $(0.1235 \times 10^{2}) + (0.2499 \times 10^{-1})$ fără a utiliza pentru rezultat forma exponenţială.

Date fiind problemele generate de diferenţa de magnitudine, este de preferat să se realizeze operaţiile asupra numerelor mici prima dată şi apoi să se combine cu cele mari. Astfel valorile mici au o şansă se devină semnificative înainte de operaţiile cu cele mari.

În mod similar, în cazul înmulţirii şi împărţirii, pentru a evita pierderile de precizie, este de preferat să se realizeze toate înmulţirile de la numărător înainte de a se împarţi la numitor. De exemplu, în cazul calculatorului nostru care rotunjeşte la 3 biţi semnificativi, calculăm expresia (x/y)z, unde x=2,41, y=9,75, z=1,54. Rezultatul lui x/y este 0,247179 şi se rotunjeşte la 0,247, şi înmulţit cu z se obţine (x/y)z = (0,247) (1,54) = 0,38038, sau 0,380 după rotunjire. In schimb deoarece d.p.d.v. matematic (x/y)z =(xz)/y, efectuând înmulţirea prima dată se obţine (xz)/y = 3,71/9,75 = 0,380513, rotunjit la 0,381. Ultima valoare este mai apropiată de cea corectă de 0,380656…

Totuşi, dacă în cazul înmulţirilor urmate de împărţiri sunt şanse să se producă erori overflow, pentru exemplu (xz)/y este de recomandat ca operaţiile să se realizeze în ordinea iniţială, adică (x/y)z.

Întrebări recapitulative 5. Presupune ca variabilele *r, u, x, y, şi z* sunt numere reprezentate cu virgula flotantă. Scrieţi următoarea expresie astfel încât să se reducă erorile date de round-off, presupunând ca nu sunt probleme de generate de overflow sau underflow:

$$\left(\frac{3}{z}\right)(r)\left(\frac{x+y}{u}\right)$$

1.2.9. Propagarea erorilor

Buclele unui program, instrucţiunile iterative care conţin operaţii aritmetice cu valori exprimate în virgulă flotantă pot determina erori de tip round-off.

Acumularea unor astfel de erori poate avea consecinţe dezastruoase. Ca şi exemplu serveşte situaţia din 1991 când în războiul din Golf sistemele antirachetă Patriot (ale USA) nu au interceptat o rachetă Scud cauzând moartea a 28 de soldaţi si rănirea a peste o sută în Dharan, Arabia Saudită. Acest dezastru s-a datorat acumulării de erori. Deoarece ceasul intern al sistemului Patriot măsura timpul în zeci de secunde, pentru a obţine valorile în secunde se înmulţea numărul de impulsuri furnizate de ceas cu 1/10. Cum rezultatele se memorau pe 24 de biţi şi operaţiile se realizau în binar sistemul nu a putut păstra în întregime valorile. Astfel fiecare zecime de secundă determina o eroare de 0,000 000 095 sec. La momentul dezastrului, sistemul patriot opera de aproximativ 100h fără întrerupere (re-boot-are) şi aceasta a cauzat un decalaj faţă de timpul real de (100hrs)(60min/hr)(60sec/min)(10 impulsuri/sec) (0.000 000 095 sec/impuls) = 0,34 sec. Pe parcursul acelei treimi de secundă racheta Scoud a zburat aproximativ 1676 metri, fiind departe de locul unde ar fi putut fi interceptată.

Exemplul 3. În multe aplicaţii, care monitorizează diferite procese la intervale constante de timp, momentul măsurării este determinat de expresii de genul $t=t+dt$, unde t apoi, din partea dreaptă, ajunge în partea stângă pentru următoarea iteraţie. De exemplu, presupunem că iteraţia este repetată de 600 de ori, spaţiul de stocare este limitat şi este nevoie şi de o conversie în baza 2. Astfel pentru *dt* care în loc să fie de 0,1 secunde este de $0,09961=0,9961 \times 10^{-1}$, pe măsura ce bucla se execută, eroarea propagată creşte. Astfel la iteraţia 11 chiar dacă

valoarea ar trebui sa fie (11)(0,09961) = 1.09571, pentru un calculator cu precizia 4, valoarea rotunjită este 1,096, eroarea absolută este de 0,00029, iar cea relativă de aproximativ 0,026%. După iteraţia 51, eroarea relativă este de 0,3128% şi este de aproximativ 30 de ori mai mare decât cea din iteraţia a doua.

Pentru a evita acumularea erorilor în buclele unor programe, *t* trebuie calculat înmulţind indexul cu valoarea lui *dt* astfel *t=i*dt*. Eroarea round-off mai există, dar efectul ei este minimizat deoarece nu se acumulează. Pentru a se compara rezultatele se poate completa un tabel şi se pot calcula valorile lui *t* pentru diferite valori ale indexului *i* utilizând ambele formule de calcul.

În general, în instrucţiuni iterative, este recomandat să se evite acumularea unor valori în virgulă flotantă prin adunări sau scăderi repetate.

Întrebări recapitulative 6. Care dintre atribuirile de mai jos sunt mai potrivite (dacă există vreo diferenţă) într-o buclă cu indexul *k* având valoarea iniţială 1. Variabila *sum* a fost iniţializată deja la 0 înainte de buclă.

a. sum = sum + 0.00492

b. sum = 0.00492 * k

c. nu contează.

1.2.10. Nerespectarea proprietăţilor numerelor

În secţiunea despre erori aritmetice am văzut că proprietăţile numerelor nu sunt îndeplinite întotdeauna în aritmetica calculatoarelor (ex. chiar dacă *(x/y)z =(xz)/y*, am văzut că cele două expresii nu generează acelaşi rezultat în unele cazuri în implementările pe calculator). Alte proprietăţi nerespectate sunt asociativitatea - adică *(a + b) + c = a + (b + c)*, şi *(ab)c = a(bc)* – respectiv distributivitatea – adică *a (b + c) = ab + ac*. În cazul calculatoarelor, dacă *a* şi *b* sunt foarte mici în comparaţie cu *c* atunci aceste proprietăţi sunt încălcate.

Întrebări recapitulative 7. Arătaţi că distributivitatea nu este respectată pentru expresiile x(y + z) şi xy + xz pentru x=2,48, y=9,34, z=1,55 când se utilizează o maşină care trunchiază rezultatul final şi cele intermediare după 2 cifre.

1.2.11. Compararea numerelor în virgulă flotantă

Datorită numeroaselor erori care pot să apară în implementarea operaţiilor cu numere reale, compararea numerelor reale trebuie evitată. Astfel, pentru a realiza aceeaşi funcţionalitate, acceptând o marjă de eroare, este de preferat să se folosească expresii de genul:

„dacă |x – z| < 0,001, considerăm că x şi z sunt identice.”, (1)

unde 0.001 este o valoare folosită pentru a exprima diferenţa acceptată dintre valori (în valoare absolută) şi faptul că sistemul foloseşte doar trei cifre semnificative pentru reprezentare.

Întrebări recapitulative 8. Scrieţi o expresie similară celei notate cu (1) pentru a compara două valori în virgulă flotantă, diferenţa acceptată fiind de 0,000 001.

1.2.12. Erori de trunchiere

Am observat deja că utilizând tehnica de calcul nu pot fi memorate corect numere de precizie infinită. Acelaşi lucru este valabil şi pentru calcularea unor expresii cu un număr infinit de termeni. Să consideram cazul numărului lui Euler *e* la puterea *x,* care este definit:

$$e^x = 1 + x + \frac{x^2}{1 \cdot 2} + \frac{x^3}{1 \cdot 2 \cdot 3} + \frac{x^4}{1 \cdot 2 \cdot 3 \cdot 4} + \cdots + \frac{x^n}{n!} + \ldots$$

Astfel, evaluarea expresiei $e^1 = e$, care are valoarea 2,718281828459045… este identică cu rezultatul evaluării expresiei de mai jos, unde *x* din expresia generală a fost înlocuit cu 1:

$$e = 1 + 1 + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \cdots + \frac{1}{n!} + \ldots$$

O maşina nu poate realiza un număr infinit de adunări de astfel de termeni. De aceea, de exemplu, după 20 de adunări se obţine doar un rezultat parţial. Astfel, termenul 1/(21!) nu este inclus în rezultat şi nici următorii, rezultând o eroare de trunchiere de aproximativ 2,05 x 10^{-20}.

Astfel, *eroarea de trunchiere* este dată de calcularea unui număr finit de operaţii în locul unui număr infinit de operaţii ale unei serii.

Întrebări recapitulative 8. Pe baza calculelor se ştie că *sin(x)* este dată de următoarea expresie:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

a. Calculaţi sin(2) pentru seria trunchiată la 2 termeni.

b. Precizaţi eroarea pentru această aproximare.

1.3. Exerciţii propuse

Scrieţi valorile de la exerciţiile 1 – 12 în formă exponenţială normalizată.

1. 63,850	2. 29,748	3. 0,00032	4. 53,7 x 10^3
5. 0,496	6. 0,0000017	7. 0,009 x 10^{-5}	8. 0,009 x 10^5
9. -0,82	10. -82	11. -0,00082	12. 4,4

13. Precizaţi magnitudinea şi precizia pentru 0,743621 x 10^{25}.

14. Precizaţi magnitudinea şi precizia pentru 93,6 x 10^7.

15. Care este precizia şi cea mai mare magnitudine care poate fi utilizată de un calculator care permite 8 cifre semnificative şi cea mai mare putere a lui 10 este 125?

Specificaţi numărul de cifre semnificative precum şi cele mai semnificative cifre pentru exerciţiile 16 – 21.

16. 63,850	17. 29,004	18. 0,00074
19. 10^3	20. 4 x 10^{-5}	21. 0,300500

22. Specificați mărimea/magnitudinea unor numere normalizate pozitive în cazul cărora parte semnificativă are 3 cifre şi exponentul lui 10 poate varia între -5 şi +5.

23. Specificați mărimea/magnitudinea unor numere normalizate pozitive în cazul cărora parte semnificativă are 7 cifre şi exponentul lui 10 poate varia între -78 şi +73.

Pentru exercițiile 24-26 determinați (a) eroarea absolută şi (b) eroarea relativă a fiecărei valori dat fiind că se foloseşte o rotunjire la două cifre zecimale. După aceea să se determine (c) eroarea absolută şi (d) eroarea relativă dacă se foloseşte trunchierea după două poziții zecimale.

24. 6,239 25. 6,231 26. 1,0/3,0 memorat cu 5 cifre semnificative

27. a. Calculați $(9{,}4 \times 10^{-5}) + (3{,}6 \times 10^{4})$. Reprezentați rezultatul în formă normalizată exponențială.

b. Rotunjiți rezultatul utilizând doar 5 cifre semnificative.

c. Calculați eroarea absolută pentru b.

d. Calculați eroarea relativă pentru b.

28. Repetați exercițiul 27 pentru expresia $(0{,}7 \times 10^{3}) - (0{,}825 \times 10^{2})$. Utilizați 3 cifre semnificative în loc de 5 pentru punctele b, c şi d.

29. Presupunem că se execută următoarea secvență de cod: *x = 6,239; x = x + x*. Pentru o maşină care trunchiază după 3 cifre semnificative, dați valorile lui *x* după fiecare instrucțiune şi eroarea relativă după ultima instrucțiune. Comparați eroarea cu rezultatul de la exercițiul 24.

30. a. Cu o aplicație de calcul evaluați următoarea expresie:

10000000000000000, + 1, -10000000000000000,

adică $1{,}0 \times 10^{16} + 1, - 1{,}0 \times 10^{16}$.

b. Care ar trebui sa fie rezultatul?

c. Explicați ce s-a întâmplat.

31. Utilizând o aplicație de calcul (de exemplu Excel) evaluați următoarele expresii pentru t = 355/113, r=101/113 şi s = 52/113: t – s – r – r – r, t – r – r – r – s, t – r – r – s – r, t – r – 2r – s, t – r – s – r – r, t – 2r – r – s, t – 3r – s, t – r – s – 2r. Utilizând cunoştințele de matematică, care sunt valorile acestor expresii? Care proprietate (proprietăți) algebrice sunt încălcate?

32. Pe baza calculelor se ştie că funcția *cos(x)* este dată de următoarea serie infinită:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

a. Calculați cos(2) pentru seria trunchiată la 3 termeni.

b. Calculați cos(2) pentru seria de mai sus până când aproximările succesive sunt identice pentru primele 4 cifre semnificative. Dați acea valoare.

2. Analiza sistemelor

2.1. Prezentarea lucrării

După reluarea unor noțiuni de la curs referitoare la clasificarea sistemelor, în cadrul acestei lucrări sunt propuse probleme de analiză a sistemelor – se cere construirea unor diagrame de timp şi compararea efortul de procesare în funcție de modelul ales – condus de timp sau de evenimente.

2.2. Clasificarea sistemelor

Prima etapă a procesului de modelare este dedicată analizei problemei/sistemului modelat. În aceasta etapă de analiză se identifică cel mai potrivit model (figura 2.1) pentru a descrie problema/sistemul analizat. Pentru a identifica cel mai potrivit model se analizează procesul – dacă este static sau dinamic, dacă depinde de timp sau nu, dacă este condus (directat) de timp sau evenimente, etc.

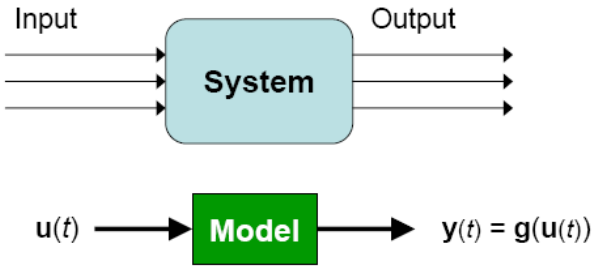


Figura 2.1. Procesul de modelare. De la sistem la model

În figura 2.2 este prezentată o clasificare a sistemelor.

Astfel, pentru modelul din figura 2.1, un sistem este considerat *static* dacă ieşirile *y(t)* sunt independente de valorile din trecut ale intrărilor $u(\tau)$ unde $\tau < t$ pentru toate valorile lui t (u şi y pot fi sub formă de vector). Un sistem *dinamic* este caracterizat prin faptul că ieşirile depind în general de valorile din trecut ale intrărilor.

Pentru a face diferența între sisteme care depind de timp şi cele care nu depind de timp e nevoie să răspundem la următoarea întrebare: *Ieşirea sistemului este identică pentru toate situațiile în care aceleaşi valori sunt aplicate pe intrări?* Deoarece răspunsul nu poate fi în general „da” trebuie să continuăm în încercarea de a clasifica sistemele. Astfel, dacă *y=g(u, t)* şi astfel funcția de ieşire depinde de variabila de timp avem de a face cu un *sistem care depinde de timp*. Altfel, dacă funcția de ieşire nu depinde de variabila de timp t discutăm despre un *sistem independent de timp*.

Conceptul de *stare* pentru un sistem descrie „comportamentul” acestuia la momentul de timp t. Astfel, dacă vectorul de intrare *u(t)* este complet specificat pentru orice moment de timp $t \geq t_0$, *starea* sistemului la momentul t_0 constă în suma informațiilor de la momentul t_0 necesare pentru a determina în mod unic ieşirea *y(t)* pentru toate momentele de timp $t \geq t_0$.

Considerând *u(t)* şi *y(t)* vectori, starea va fi de asemenea un vector pe care îl notăm cu *x(t)*. Componentele vectorului *x(t)*, adică $x_1(t)$, $x_2(t)$ etc. se numesc *variabile de stare*.

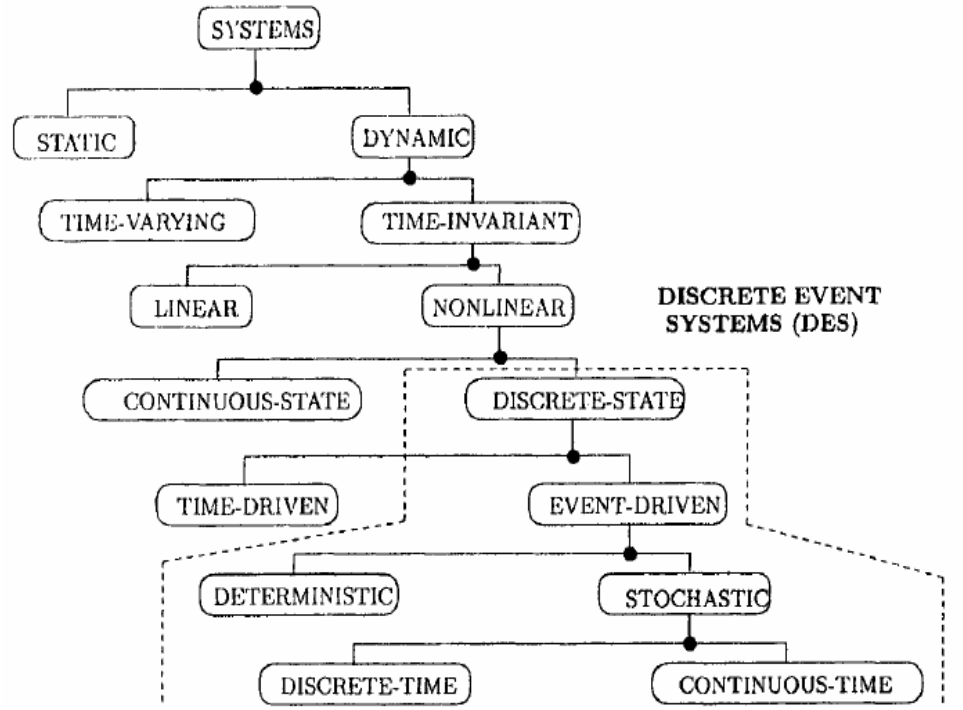


Figura 2.2. Clasificarea sistemelor[1]

Setul de ecuaţii necesare pentru a descrie stările *x(t)* pentru toate momentele de timp $t \geq t_0$ când se cunosc $x(t_0)$ şi funcţia *u(t)*, pentru toate momentele de timp $t \geq t_0$, se numesc *ecuaţii de stare*. *Spaţiul stărilor* pentru un sistem, de obicei notat cu X, este format din totalitatea stărilor pe care sistemul le poate avea. Astfel, pentru ecuaţiile de stare putem avea următoarea expresie diferenţială: $\dot{x}(t) = f(x(t), u(t), t)$, unde $x(t_0)=x_0$ (figura 2.3).

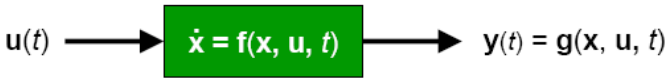


Figura 2.3. Intrările, ecuaţiile de stare şi funcţiile de ieşire

În funcţie de forma funcţiilor *f* şi *g*, utilizate pentru a determina starea următoare a sistemului respectiv ieşirile sistemului, putem să clasificăm sistemul ca fiind linear sau neliniar. O funcţie *g* spunem că este *liniară* dacă şi numai dacă pentru funcţia $g : U \rightarrow Y$, cu *u* şi *v* aparţinând lui *U*, avem $g(au+bv)=ag(u)+bg(v)$ pentru oricare numere reale *a* şi *b*. Astfel, despre un *sistem* spunem că este *linear* dacă şi numai dacă ambele funcţii *f* şi *g* sunt funcţii lineare.

[1] *Introduction to Discrete Event Systems, second edition*, Christos G. Cassandras & Stephane Lafortune, Springer, 2008

Pornind de la spaţiul stărilor unui sistem se mai poate face o clasificare. Modelul poate avea *spaţiul stărilor continuu*, de exemplu cazul în care X poate lua orice valoare reală (figura 2.4) sau poate avea *spaţiul stărilor discret*, de exemplu cazul în care X poate lua doar valori întregi sau naturale (numărul de obiecte de inventar dintr-un depozit este întotdeauna întreg).

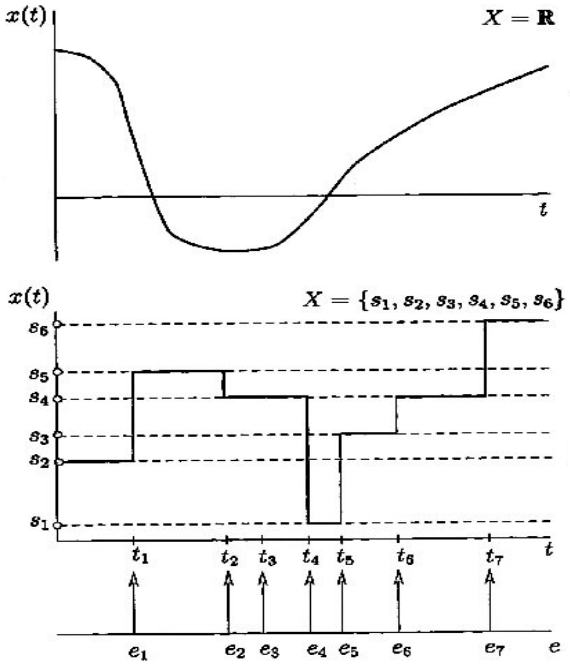


Figura 2.4. Spaţiul stărilor. Spaţiu continuu vs. spaţiu discret[1]

Un sistem este *stohastic* (sau probabilistic) dacă cel puţin o variabilă de ieşire este aleatoare. Altfel, sistemul este *determinist*.

Dacă pentru un sistem sunt definite variabilele de intrare şi cele de ieşire doar la intervale discrete de timp, sistemul se numeşte *sistem cu timp discret*, altfel el se numeşte *sistem cu timp continuu*.

În plus, dacă tranziţia de la o stare la alta este dată de evenimente, vorbim despre *sisteme directate (conduse) de evenimente* spre deosebire de sistemele *directate de timp*, unde tranziţiile sunt sincronizate cu o temporizare (cu un ceas care e responsabil de declanşarea tranziţiilor). Este mai greu de modelat sistemele directate de evenimente; evenimentele pot fi comparate prin analogie cu întreruperile pentru sistemele de calcul.

2.3. Sisteme cu evenimente discrete (DES)

Prin *Sistem cu Evenimente Discrete (DES – Discrete Event System)* înţelegem un sistem care îndeplineşte simultan următoarele condiţii:

(a) spaţiul stărilor este o mulţime discretă iar

(b) mecanismul de declanşare a tranziţiilor este determinat de evenimente.

Deci, DES este un sistem directat de eveniment şi care are un spaţiul al stărilor discret. Evoluţia de la o stare la alta se realizează doar la apariţia unui eveniment (ex. stările unei

maşini pot fi ex. {ON, OFF}; {IDLE, BUSY, DOWN}, şi stările la rândul lor pot fi divizate iar evenimentele pot fi apăsarea unui buton, selectarea unei comenzi, etc.).

2.4. Sisteme cu cozi de aşteptare (queueing systems)

Sunt trei elemente de bază care definesc o coadă de aşteptare:

- entităţile care 'aşteaptă', sau *clienţii* (oameni în staţia de autobus, mesaje în mediul de comunicare, tranzacţii executate pe PC, produse parţial finisate, maşini pe şosea, avioane la decolare…);
- resursele pentru care se aşteaptă, sau *servere* (autobuze, vânzători în magazin, canale de comunicare, procesoare, periferice, utilaje, semafoare…
- spaţiul de aşteptare, *şirul sau coada de aşteptare* (staţii de aşteptare, depozite, buffer-e…).

Motivaţia utilizării sistemelor cu cozi de aşteptare e dată de limitarea resurselor care necesită alocarea acestora în mod echitabil. Astfel e nevoie ca (a) resursele să fie corect şi eficient alocate pentru toţi clienţii, (b) clienţii să fie serviţi, (c) costul de proiectare şi mentenanţa să fie acceptabil. Sistemele cu cozi de aşteptare sunt caracterizate prin lungimea cozii (lungimea şirului – numărul maxim de clienţi în aşteptare) şi regula cozii – ex. FIFO etc.

În figura 2.5 este reprezentat un sistem, care procesează componente, cu două servere (1 şi 2) şi cu două cozi de aşteptare, cea din partea stângă de capacitate infinită iar cea dintre servere cu un buffer pentru 2 componente (a doua coadă având capacitatea 3, capacitatea fiind dată inclusiv de componenta în lucru la serverul 2).

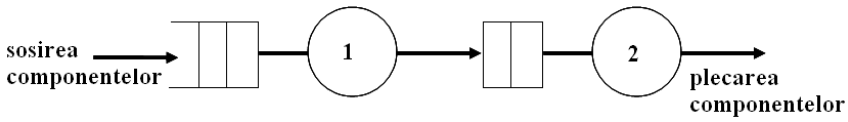


Figura 2.5. Sistem cu coadă de aşteptare

În această lucrare, cu ajutorul unor probleme, vom evidenţia diferenţă dintre procesele conduse de evenimente şi cele conduse de timp.

2.5. Probleme propuse

1. Un sistem de calcul dispune de două procesoare P_1 respectiv P_2 care funcţionează în paralel. Procesul prin care se atribuie task-urile poate fi descris în timp discret astfel: *a(t), t=0, 1, 2, ...* , unde *a(t)=1* dacă este trimis un task la momentul *t,* respectiv, *a(t)=0* dacă nu s-a trimis nici un task la momentul *t* respectiv. Presupunem ca procesul este specificat pentru intervalul *t=0, 1, ..., 10* astfel: {1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1}. Sistemul de calcul foloseşte următoarea regulă pentru atribuirea task-ului sosit: alternează între cele două procesoare, primul task fiind atribuit procesorului P_1 (cel de-al doilea pentru P_2, următorul pentru P_1, etc.). Se presupune că dacă un task soseşte la oricare dintre procesoare şi procesorul este ocupat, task-ul va sta în şirul/coada de aşteptare a procesorului (cozile de aşteptare au capacitate infinită). Timpul de procesare al lui P_1 alternează între 4 unităţi de timp şi o unitate de timp (4 unităţi de timp pentru primul task executat, o unitate de timp pentru al doilea task, etc.), în timp ce pentru procesorul P_2 timpul de procesare este de 2 unităţi de timp şi rămâne constant.

Fie *y(t)* numărul total de clienţi care au plecat de la sistem la momentul *t*, si $x_1(t)$ respectiv $x_2(t)$ lungimile şirurilor de aşteptare ale celor două procesoare P_1 respectiv P_2 (inclusiv task-ul

în lucru). Dacă mai multe evenimente se produc la momentul *t* valoarea acestor variabile este determinată chiar imediat după ce au loc evenimentele.

(a) Desenaţi o diagramă de timp pentru intervalul *t=0, 1, ..., 10* indicând exact sosirile (s) şi plecările (p), ştiind că la momentul iniţial, chiar înainte de sosirea primului task pentru procesare, valorile cozilor de aşteptare sunt $x_1(0)= x_2(0)= y(0)=0$.

(b) Construiţi un tabel cu valorile pentru $x_1(t)$, $x_2(t)$ si $y(t)$ pentru intervalul *t=0, 1, ..., 10*.

(c) Presupunem de acum încolo că se lucrează pe domeniul de timp continuu. Sosirile au loc la momentele de timp 0.1, 0.7, 2.2, 5.2 si 9.9. Timpul de procesare pentru P_1 alternează între 4.2 şi 1.1 în timp ce pentru procesul P_2 timpul de procesare este de 2.0 unităţi de timp. Considerăm un model condus de evenimente cu mulţimea evenimentelor $E=\{s, p_1, p_2\}$ unde *s* reprezintă sosirea unui task, p_i reprezintă plecări de la P_i (i=1,2). Construiţi un tabel cu valorile lui $x_1(k)$, $x_2(k)$, $y(k)$ si $t(k)$ unde $x_1(k)$, $x_2(k)$, $y(k)$ sunt lungimile şirurilor de aşteptare ale celor două procesoare P_1, P_2 si numărul cumulat de plecări după ce a avut loc evenimentul *k*, iar $t(k)$ este momentul în care evenimentul *k* a avut loc. Dacă au loc două evenimente în acelaşi timp, presupunem că plecarea are loc înaintea sosirii. Comparaţi numărul de actualizări necesare pentru acest model comparativ cu cel condus de timp cu paşi de magnitudinea 0.1 unităţi de timp.

2. Reluaţi problema 1 ţinând cont de câte una din următoarele două reguli de alocare a task-urilor (se păstrează presupunerea că dacă au loc două evenimente în acelaşi timp, plecarea are loc înaintea sosirii):

(a) Task-urile se alocă procesorului P_1 atât timp cât şirul lui P_1 de aşteptare este mai mic sau egal cu 2, altfel task-ul este alocat lui P_2.

(b) Alocarea se face procesorului cu cel mai scurt şir de aşteptare, în caz de egalitate task-ul este alocat lui P_2.

3. Un proces simplu de producţie presupune existentă a două maşini M_1 si M_2 şi a unui robot care mută componentele finalizate de M_1 la M_2. Pentru nici una dintre maşini nu există buffer. Astfel, dacă o componentă este adusă la M_1 cât timp aceasta este ocupată, componenta este respinsă. Pe de altă parte, dacă robotul transportă o piesa la M_2 în timp ce M_2 este ocupată, robotul aşteaptă până când M_2 poate prelua componentă. După ce componenta este preluată de M_2, robotul are nevoie de timp pentru a putea prelua o altă piesă de la maşina M_1. Astfel, M_1 poate fi forţată să păstreze o componentă finalizată şi să nu preia altă componentă în lucru până când robotul este din nou disponibil.

Stările lui M_1, M_2 si respectiv a robotului sunt descrise de x_1, x_2, şi x_3. Presupunând că timpul de procesare a M_1 este 0.5s, a M_2 este 1.5s şi robotul are nevoie de 0.2s să transporte o piesă de la M_1 la M_2 respectiv are nevoie de 0.1s să se întoarcă de la M_2 la M_1. De asemenea presupunem că sosirea componentelor la M_1 este programată la 0.1, 0.7, 1.1, 1.6 şi 2.5s.

(a) Identificaţi toate valorile posibile (stările posibile) pe care le pot avea x_1, x_2 si x_3.

(b) Definiţi mulţimea evenimentelor *E* (cu numărul cel mai mic de evenimente) pentru sistem.

(c) Pentru intervalul [0.0, 3.0]s construiţi un tabel cu valorile lui $x_1(k)$, $x_2(k)$, $x_3(k)$ şi $t(k)$, unde $x_1(k)$, $x_2(k)$, şi $x_3(k)$ sunt stările maşinilor M_1, M_2 şi a robotului după ce evenimentul *k* a

avut loc, *k=1, 2, . . .,* şi *t(k)* este momentul în care evenimentul *k* a avut loc. Dacă două evenimente au loc simultan, presupunem ca întotdeauna prima dată se finalizează o componentă şi apoi vine o alta nouă.

(d) Identificați toate stările în care maşina M_1 este forțată să aştepte până când robotul îndepărtează o componentă finalizată de maşina M_1.

3. Modele netemporizate pentru modelarea DES. Expresii regulate şi automate de stări

3.1. Despre lucrare

Pe parcursul acestei lucrări vor fi reluate unele noţiuni de la curs referitoare la modele independente de timp care pot fi utilizate în modelarea *sistemelor cu evenimente discrete (Discrete Event Systems - DES)*. În cadrul acestei lucrări sunt propuse probleme care utilizează teoria automatelor cu stări finite iar în lucrarea următoare vor fi utilizate reţele Petri.

3.2. Noţiuni utilizate. Expresii regulate şi automate

Din teoria automatelor cu stări finite pentru modelarea netemporizată a DES se utilizează următoarele noţiuni:

- Dacă notăm cu *E* mulţimea *evenimentelor* asociate unui proces/sistem şi considerăm această mulţime un *alfabet*, atunci secvenţe de evenimente formează *cuvinte* (şiruri). O *secvenţă* de cuvinte formate pe baza alfabetului se numeşte *limbaj*. Un *limbaj regulat* se construieşte utilizând operaţii asociate şirurilor: concatenare (marcată prin alăturare, fără operator), reuniune (notată cu +), operatorul * al lui Kleene (notat cu A^* şi definit mai jos, unde ε este şirul vid).

- $A^* = \bigcup_{n=0}^{\infty} A^n$, *unde* $A^0 = \{\varepsilon\}$, *si* $A^n = AA^{n-1}$, *pentru toate valorile* $n \geq 1$

- Pentru descrierea şirurilor de evenimente se folosesc *expresii regulate* şi operaţiile asociate şirurilor precizate mai sus (ex.: *concatenarea* evenimentelor α şi β se notează cu $\alpha\beta$, *reuniunea* evenimentelor α şi β se notează cu $\alpha+\beta$, şi *operatorul* * al lui Kleene α^*, etc).

- Un *automat cu stări finite* este un dispozitiv capabil să genereze un limbaj pe baza unor reguli. Este definit de

- mulţimea evenimentelor (alfabet finit) *E*,
- mulţimea finită a stărilor *X*,
- o funcţie de tranziţie *f*,
- starea iniţială x_0, şi
- *F* mulţimea stărilor finale.

Un şir este recunoscut de un automat cu stări finite *(E, X, f, x_0, F)*, dacă cauzează o secvenţă de tranziţii de la x_0 la o stare finală *x* din *F*. Toate şirurile cu aceste proprietăţi formează limbajul recunoscut de *(E, X, f, x_0, F)*. Există întotdeauna un automat cu stări finite care să genereze un limbaj regulat.

- În figura 3.1 este reprezentată diagrama de tranziţie a stărilor pentru un automat cu stări finite.

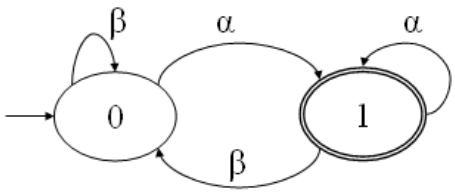


Figura 3.1. Automat cu stări finite. Exemplu

Pentru automatul din figura 3.1 elementele setului *(E, X, f, x_0, F)* care descriu automatul cu stări finite sunt:

E={α, β},

X ={0, 1},

f(0, α) = 1, f(0, β) = 0, f(1, α) = 1, f(1, β) = 0

x_0 *=0* (starea iniţială marcată cu o săgeată),

F={1} (În cazul acesta starea 1 este dublu încercuită, pentru a fi identificată ca şi stare finală),

Expresia regulată care descrie diagrama din figura 3.1 este: (α+β)*α. Această expresie reprezintă limbajul L={α, αα, βα, ααα, αβα, βαα, ββα …}, adică toate şirurile în care după α sau β urmează întotdeauna α.

- Modelul unui automat cu stări finite poate fi restrâns utilizând *agregarea* stărilor echivalente şi înlocuirea lor cu una echivalentă, fără a se pierde nici o informaţie. Astfel, dacă acelaşi şir oarecare aplicat atât din starea *x* cât şi din *y* întotdeauna duce la mulţimea de stări finale *F* atunci *x* şi *y* pot fi considerate stări echivalente. Trebuie observat că (1) două stări, una finală şi una nu – nu vor putea fi echivalente şi (2) dacă *f(x,e) =y* şi *f(y,e)=x* pt. orice eveniment *e*, atunci *x* şi *y* echivalente.

- Pentru a folosi un automat ca şi model independent de timp pentru modelarea DES presupunem că *E* şi *X* sunt mulţimi numărabile, eliminăm specificaţiile de mulţime finită pentru *F* şi introducem definirea unor mulţimi de evenimente posibile *Γ(x)* pentru fiecare stare *x*. Modelul rezultat *(E, X, Γ, f, x_0, F)* este utilizat cu denumirea *automat de stare*. Setul care descrie un automat de stare este format din:

E mulţimea de evenimente (numărabile),

X spaţiul stărilor (numărabile),

Γ(x) mulţimea evenimentelor 'posibile' din starea *x, $x \in X$, $\Gamma \subseteq E$*

f funcţia de tranziţie *f: $X \times E \rightarrow X$*, definită în *x* doar pentru *$e \in \Gamma(x)$*

x_0 starea iniţială, $x_0 \in X$.

3.3. Probleme propuse

1. Fie alfabetul E= {α, β, γ}. Găsiţi cel puţin o

(a) expresie regulată a limbajului L pentru care fiecare şir conţine cel puţin un β.

(b) expresie regulată a limbajului L pentru care fiecare şir conţine exact doi β.

(c) expresie regulată a limbajului L pentru care fiecare şir conţine cel puţin doi β.

(d) încă două expresii diferite pentru (c).

2. Fie alfabetul E= {α, β}. Pentru fiecare din perechile care urmează evaluaţi echivalenţa expresiilor. Astfel, fie demonstraţi că reprezintă acelaşi limbaj fie, în caz contrar, indicaţi un cuvânt care aparţine unui limbaj şi nu aparţine celuilalt:

(a) (α + β)*, (α + β)* αβ (α + β)* + β*α*.

(b) (α * + β)*, (α + β)*.

(c) (αβ)* αβ, β(α + β)* αβ.

(d) (α*β)*α *, α *(βα *)* .

3. Fie trei limbaje L1, L2, L3, cu L1 care nu conţine şirul vid. Arătaţi că dacă L2 = L1L2 ∪ L3, atunci L2 = L1* L3.

4. Pe baza alfabetului E= {α, β, γ} să se construiască câte un automat cu stări finite care să recunoască:

(a) limbajul {αβγ, αββ, βγγ}.

(b) limbajul reprezentat prin (αβ)*γ.

(c) limbajul reprezentat prin α(βα)*α.

5. Pornind de la alfabetul limbii engleze să se construiască un automat cu stări finite, sau un automat de stare, care să recunoască cuvintele *man* şi *woman*. Identificaţi toate elementele setului care descriu automatul cu stări finite respectiv automatul de stare reprezentat.

6. Găsiţi expresia regulată pentru limbajul recunoscut de următorul automat cu stări finite.

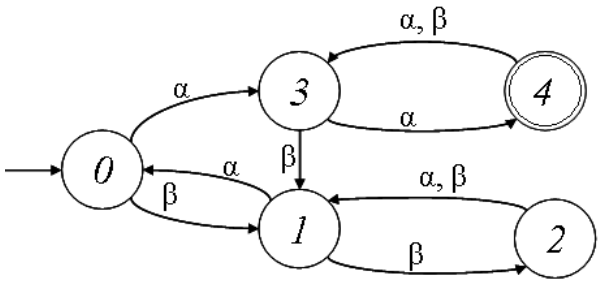


Figura 3.2 . Automatul cu stări finite pentru problema 6.

7. Un cifru pentru un seif este proiectat utilizând doar cifrele 0 şi 1. Combinaţia care deschide seiful constă din exact 4 cifre, dintre care ultimele 2 identice (doi de 1, sau doi de 0).

Dacă combinaţia nu este corectă se declanşează alarma. Construiţi un automat cu stări finite care să descrie procesul de deschidere a seifului.

8. Un sistem de calcul utilizează două procesoare P1 respectiv P2 care funcţionează în paralel. Lungimea totală a cozilor de aşteptare (incluzând elementul procesat de server) pentru P1 este K1=1, iar pentru P2 este K2=2. Sistemul primeşte două tipuri de job-uri: J1 şi J2. Job-urile de tip J1 sunt procesate la P1 şi cele de tip J2 trebuie procesate la P2. Când un job este procesat el părăseşte sistemul. Dacă coada de aşteptare este plină la sosirea unui job, atunci acel job este respins. Lungimea cozilor este notată cu x1 şi respectiv x2. Definiţi un automat de stare (E, X, Γ, f, x_0, F) pentru sistem şi construiţi diagrama stărilor indicând toate evenimentele posibile.

4.Rețele Petri. Modele netemporizate pentru DES

4.1. Prezentarea lucrării

Pe parcursul acestei lucrări vor fi reluate unele noțiuni de la curs referitoare la rețelele Petri care pot fi utilizate pentru modelarea netemporizată a *sistemelor cu evenimente discrete (DES)*. La finalul lucrării sunt propuse probleme utilizând rețele Petri.

4.2. Rețele Petri netemporizate. Rezumatul noțiunilor utilizate

- *Rețelele Petri* pot fi folosite pentru modelarea netemporizată a DES. Pentru definirea Rețelelor Petri (RP) se utilizează:

– mulțimea finită a *locațiilor* (places) $p_i \in P$,

– mulțimea finită a *tranzițiilor* (transitions) $t_j \in T$,

– mulțimea arcurilor orientate A, submulțime a mulțimii $(P \times T) \cup (T \times P)$ şi

– o funcție de greutate (pondere) $w(p_i, t_j)$ sau $w(t_j, p_i)$ asociată fiecărui arc; funcția w fiind definită astfel $w: A \rightarrow \{1, 2, 3, \ldots\}$.

Tranzițiile corespund *evenimentelor*, iar locațiile de intrare ale unei tranziții sunt asociate *condițiilor* care trebuie îndeplinite pentru ca tranziția/evenimentul să aibă loc.

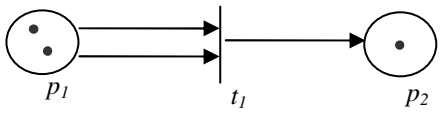


Figura 4.1. Exemplu de rețea Petri marcată.

Pentru rețeaua Petri din figura 4.1 se pot identifica:

– locații (condiții)	$P = \{p_1, p_2\}$
– tranziții (evenimente)	$T = \{t_1\}$
– arcuri	$A = \{(p_1,t_1), (t_1,p_2)\}$
– ponderi (greutăți)	$w(p_1,t_1)=2$ şi $w(t_1,p_2)=1$
– intrări / ieşiri	$I(t_1)= \{p_1\}$ şi $O(t_1)= \{p_2\}$

- Comportamentul dinamic al rețelelor Petri este descris cu ajutorul unor *jetoane* sau *marcaje* (token) plasate în locațiile care permit realizarea tranzițiilor (locațiile p_1 şi p_2 din figura 4.1 au două respectiv un jeton). Activarea şi apoi declanşarea unei tranziții cauzează „mutarea" jetoanelor. Starea unei rețele Petri este descrisă de un vector $[x(p_1), x(p_2), \ldots, x(p_n)]$, unde $x(p_i)$ este numărul de jetoane prezente în locația p_i.

Funcția de marcare x este definită $x : P \rightarrow \{0, 1, 2, \ldots\}$.

Astfel pentru reţeaua Petri din figura 4.1 starea reprezentată este dată de numărul de jetoane din locaţii: *x=[2, 1]*, valoarea 2 pentru locaţia p_1 iar valoarea 1 pentru locaţia p_2.

- O reţea Petri marcată este descrisă de setul (P, T, A, w, x_0), unde

– *P* – mulţimea finită a locaţiilor, $p_i \in P$

– *T* – mulţimea finită a tranziţiilor, $t_i \in T$

– *A* – mulţimea arcurilor, submulţime a mulţimii $(P \times T) \cup (T \times P)$

– *w* – funcţia de pondere $w: A \rightarrow \{1, 2, 3, \ldots\}$

– x_0 – marcarea iniţială.

- O tranziţie este *validată* dacă $x(p_i) \geq w(p_i, t_j)$ (numărul de jetoane a locaţiei de intrare este mai mare sau egal decât ponderea conexiunii de la locaţie la tranziţia analizată pentru toate locaţiile de intrare ale tranziţiei analizate) pentru toate $p_i \in I(t_i)$. În reţeaua din figura 4.1 tranziţia t_1 este validată deoarece în locaţia p_1 (singura locaţie de intrare) sunt cel puţin 2 jetoane (cel puţin ponderea conexiunii (p_i, t_j)).

- Când o tranziţie este validată, ea poate să se *declanşeze* (fire), ceea ce determină schimbarea marcajului (şi implicit a stării).

- Pentru o reţea Petri descrisă de (P, T, A, w, x_0) se defineşte *funcţia de tranziţie a stărilor f* pentru tranziţia $t_j \in T$, $f: \{1, 2, \ldots\}^n \times T \rightarrow \{1, 2, \ldots\}^n$ dacă şi numai dacă

(a) $x(p_i) \geq w(p_i, t_j)$ pentru toate locaţiile $p_i \in I(t_j)$ şi

(b) dacă $f(x, t_j)$ este definită, atunci stabilim $x' = f(x, t_j)$ unde

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i) \text{ , } i=1, \ldots, n.$$

- Este întotdeauna posibil să se obţină o reţea Petri dintr-un automat cu stări finite. Deşi reţelele Petri sunt modele cu care se lucrează mai greu, datorită complexităţii, ele pot fi utilizate pentru modelarea unor DES mult mai generale/ample.

- *Arborele de acoperire* poate fi utilizat pentru rezolvarea multor problemelor funcţionale de bază legate de modelarea independentă de timp a DES, cum ar fi: acoperirea, delimitarea şi conservarea. Pentru un DES cu mulţimea stărilor finită, arborele de acoperire devine arbore de accesibilitate.

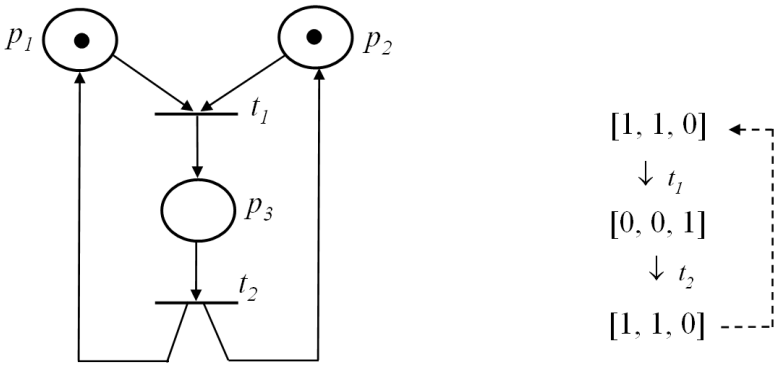


Figura 4.2. Exemplu de reţea Petri marcată.

Figurii 4.2 prezintă graful unei rețele Petri (partea stângă) şi arborele de acoperire corespunzător (în partea dreaptă).

Pentru descrierea *arborelui de acoperire* se utilizează următoarele noțiuni:

– *nod rădăcină* - primul nod, corespunzător stării inițiale

– *nod terminal* - nod din care nici o tranziție nu mai poate fi declanşată

– *nod duplicat* - identic cu un altul existent în arbore (spre care duce linia întreruptă)

– *dominanța nodurilor* - dacă x=[x(p_1), x(p_2), …, x(p_n)] şi y=[y(p_1), y(p_2), …, y(p_n)] sunt 2 stări, noduri ale arborelui de acoperire, x *domină pe* y (scris x $>_d$ y) dacă sunt îndeplinite condițiile:

(a) $x(p_i) \geq y(p_i)$, pentru toți i=1, …, n

(b) $x(p_i) > y(p_i)$, măcar pentru câțiva i=1, …, n

De exemplu [1, 0, 2, 0] >d [1, 0, 1, 0], deoarece locația a treia în prima stare (dominantă), prin numărul de jetoane 2, domină a doua stare care are tot pentru locația a treia doar un jeton.

– *simbolul* ω, marcaj pentru infinit într-o rețea nemărginită, [1, 0, 1, 0] $\rightarrow$ [1, 0, ω, 0], când marcajul ar creşte la infinit.

- Algoritmul de realizare a arborelui de acoperire este descris în continuare:

– Pasul 1 – se inițializează $x=x_0$ (starea inițială),

– Pasul 2 – pentru fiecare nod nou se evaluează funcția de tranziție $f(x,t_j)$ pentru toți $t_j \in T$,

– Pasul 2.1 – dacă $f(x,t_j)$ este nedefinită pentru toți $t_j \in T$ (nici o tranziție nu e validată) atunci x e nod terminal,

– Pasul 2.2 – dacă $f(x,t_j)$ este definită pentru unii $t_j \in T$ atunci se creează nod nou pentru $x'= f(x,t_j)$ apoi,

– Pasul 2.2.1 – dacă $x(p_i)=\omega$ pentru unele locații p_i, se atribuie pentru acele locații $x'(p_i)=\omega$,

– Pasul 2.2.2 – dacă există un nod y pe calea de la nodul x_0 la x astfel încât $x' >_d y$ se atribuie $x'(p_i)=\omega$ pentru toate locațiile p_i la care $x'(p_i) >_d y(p_i)$,

– Pasul 2.2.3 – altfel se atribuie $x'= f(x,t_j)$,

– Pasul 3 – dacă toate nodurile noi sunt ori duplicate ori terminale, stop.

Astfel rezultă un arbore finit de acoperire pentru rețeaua Petri.

- O rețea Petri se numeşte *viabilă* (live(-ness)) dacă indiferent de marcajul care a fost atins din x_0, este posibil ca, în continuare, să se execute orice tranziție a rețelei.

- O rețea Petri pentru a fi *mărginită* nu trebuie să conțină ω, altfel rețeaua are un spațiu infinit al stărilor.

- O rețea Petri este *persistentă* dacă, pentru oricare 2 tranziții validate, faptul că una se declanşează nu determină invalidarea celei de a doua (dacă două tranziții sunt validate de acelaşi set de condiții).

- O stare x este *accesibilă* (reachable) într-o rețea Petri din x_0 dacă există o secvență de tranziții care încep în x_0 astfel încât starea rețelei să devină la un moment dat x.

- O stare *y* este *acoperită* (coverable) pornind din x_0, dacă există o secvență de tranziții care încep în x_0 astfel încât starea devine la un moment dat *x* şi în plus $x(p_i) \geq y(p_i)$ pentru toate $i=1, \ldots, n$ (adică permite şi validarea tranzițiilor din acea stare).

4.3. Probleme propuse

1. Fie o rețea Petri descrisă de:

$P=\{p_1, p_2, p_3, p_4\}$,

$T=\{t_1, t_2, t_3\}$,

$A= \{(p_1, t_1), (p_1, t_3), (p_2, t_2), (p_3, t_2), (p_3, t_3), (p_4, t_3), (t_1, p_2), (t_1, p_3), (t_2, p_2), (t_2, p_4)\}$,

iar ponderile asociate acestor arcuri sunt 1.

(a) Reprezentați rețeaua Petri asociată.

(b) Fie $x_0=[2, 0, 0, 1]$ starea inițială. Identificați pentru fiecare dintre tranzițiile rețelei dacă sunt validate (dacă se pot declanşa) şi dacă da, indicați care ar fi stările următoare.

(c) Reprezentați arborele de acoperire pentru această rețea.

2. Fie o rețea Petri descrisă de:

$P=\{p_1, p_2, p_3\}$,

$T=\{t_1, t_2, t_3\}$,

$A= \{(p_1, t_1), (p_1, t_3), (p_2, t_1), (p_2, t_2), (p_3, t_3), (t_1, p_2), (t_1, p_3), (t_2, p_3), (t_3, p_1), (t_3, p_2)\}$,

iar ponderile asociate acestor arcuri sunt 1, excepție $w(p_1, t_1)=2$.

(a) reprezentați rețeaua Petri asociată.

(b) Fie $x_0=[3, 2, 1]$ starea inițială. Identificați pentru fiecare dintre tranzițiile rețelei dacă sunt validate şi dacă da, indicați care ar fi starea următoare.

(c) Fie $x_0=[1, 0, 1]$ starea inițială. Arătați că în nici o operație care urmează în rețeaua Petri tranziția t_1 nu poate avea loc.

(d) Fie $x_0=[2, 1, 1]$ o altă starea inițială. Arătați că în paşii care urmează în rețeaua Petri ori apare un blocaj (adică nu se mai poate realiza nici o tranziție) ori se revine în starea inițială x_0.

(e) Pentru punctul (d) reprezentați arborele de acoperire şi analizați arborele pentru aceleaşi demonstrații.

3. Fie rețeaua Petri de mai jos cu marcajul inițial $x_0=[1, 1, 0, 2]$.

(a) După ce se declanşează două tranziții, găsiți o stare pentru care toate tranzițiile sunt moarte.

(b) Presupunem că dorim să aplicăm următoarea secvență de declanşare ($t_3, t_1, t_3, t_1, \ldots$). Arătați că nu se poate repeta această secvență la infinit.

(c) Identificați starea x_s rezultată în urma secvenței de declanşare (t_1, t_2, t_3, t_3, t_3).

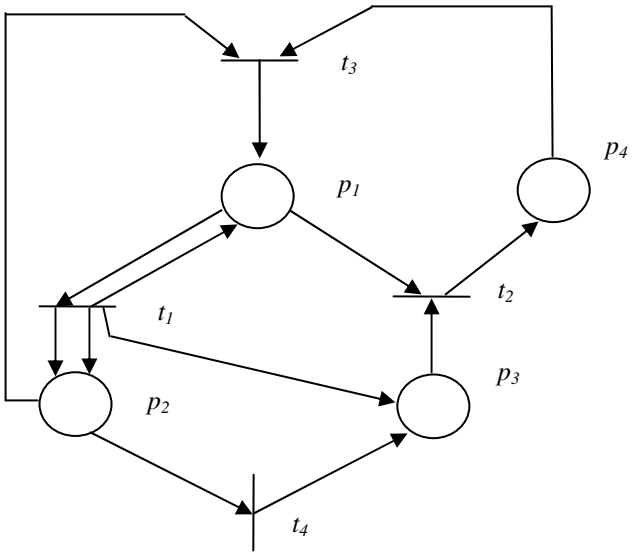


Figura 4.3. Rețeaua Petri pentru problema 3.

(a) După ce se declanşează două tranziții, găsiți o stare pentru care toate tranzițiile sunt moarte.

(b) Presupunem că dorim să aplicăm următoarea secvență de declanşare (t_3, t_1, t_3, t_1, …). Arătați că nu se poate repeta această secvență la infinit.

(c) Identificaţi starea x_s rezultată în urma secvenței de declanşare (t_1, t_2, t_3, t_3, t_3).

4. Pentru diagrama de tranziție de mai jos (figura 4.4), cu E={e_1, e_2}, desenați rețeaua Petri corespunzătoare. Apoi

(a) indicați un posibil marcaj pentru fiecare din cele 6 stări.
(b) există stări pentru care nici o tranziție nu este validată?

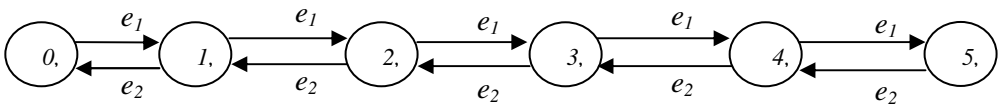


Figura 4.4. Diagrama de tranziție a stărilor pentru problema 4.

4.4. Utilizarea aplicației PIPE (Platform Independent Petri Net Editor)

În această secțiune, după ce studenții s-au familiarizat teoretic cu dinamica rețelelor Petri, urmează să utilizeze, pentru verificare, o aplicație care permite vizualizarea schimbărilor de stare în rețelele Petri.

Pentru aceasta s-a ales o aplicație Open Source[1] care permite rularea pe diferite sisteme de operare.

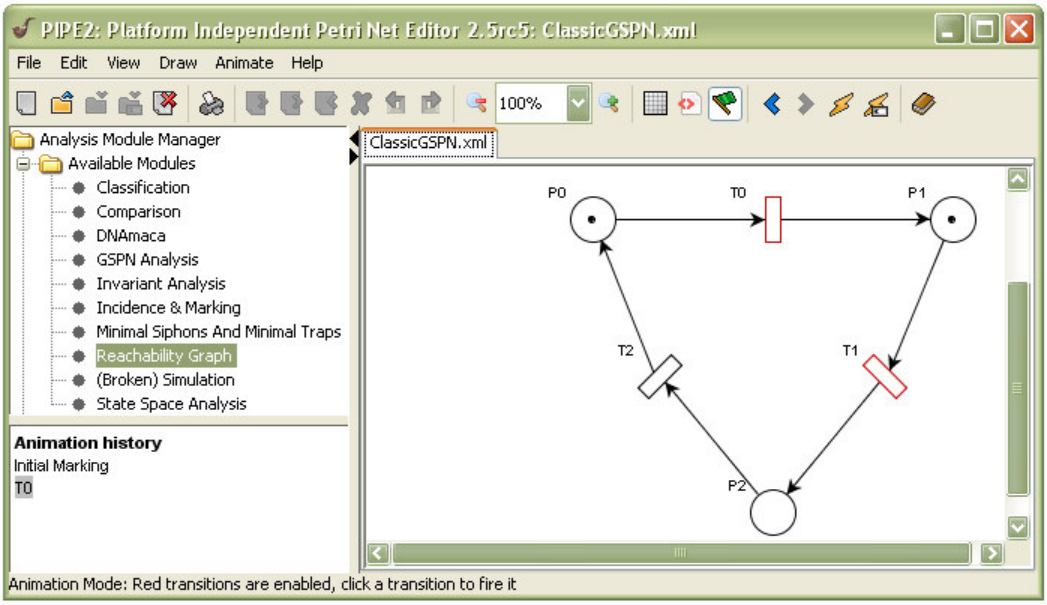


Figura 4.5. PIPE2 (Platform Independent Petri Net Editor)

În figura 4.5 este prezentată fereastra aplicației PIPE cu un exemplu de reprezentare, exemplu dintre cele disponibile în cadrul aplicației. (În acest exemplu tranzițiile din figură sunt temporizate, fiind reprezentate cu ajutorul unor dreptunghiuri, iar cele colorate în roşu sunt validate. În partea de jos, stânga este indicat istoricul tranzițiilor, pentru exemplul dat după marcajul inițial a avut loc o declanşare a tranziției T0).

Aplicația PIPE permite reprezentarea rețelelor Petri temporizate (care sunt discutate în alte lucrări) dar şi a celor netemporizate. Pentru această lucrare se folosesc doar rețelele netemporizate. Aplicația permite reprezentarea unei rețele Petri şi apoi, poate fi vizualizată dinamica acesteia. Astfel se pot plasa jetoane în locațiile reprezentate şi pe baza acestor marcaje tranzițiile validate îşi schimbă culoarea. Cu un click pe oricare dintre tranziții validate se declanşează tranziția indicată.

Aplicația permite generarea matricei de incidență şi de asemenea a grafului (arborelui) de acoperire cum se poate vedea în exemplul din figura 4.6. Pentru fiecare nod al grafului se poate vedea care sunt nodurile prin care putem ajunge în acea stare, respectiv, care sunt stările care pot urma – în funcție de tranziția declanşată.

[1] Aplicația *PIPE2 (Platform Independent Petri Net Editor)* este disponibilă (pagina a fost ultima dată accesată în noiembrie 2008) pe http://pipe2.sourceforge.net/

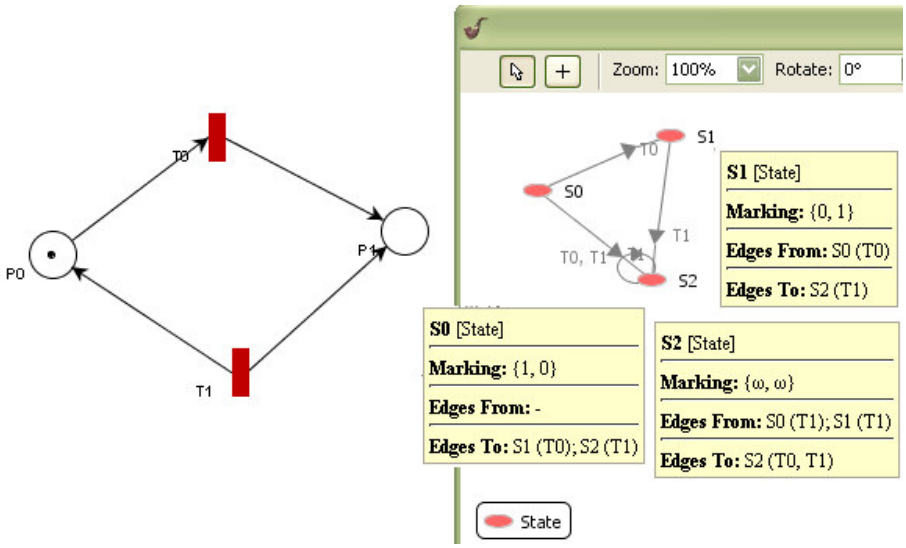


Figura 4.6. PIPE permite reprezentarea arborelui (graf-ului) de acoperire

Astfel starea S0 este cea care are marcajul inițial [1,0] şi validează ambele tranziții T0 şi T1 (de fapt T1 este întotdeauna validată). Din starea S0 se poate ajunge în starea S1 dacă se declanşează tranziția T0 respectiv în starea S2 dacă se declanşează tranziția T1, după cum se poate vedea şi din descrierea asociată stării S0. În plus se poate vedea că starea S2 are marcajul [ω, ω], deoarece declanşarea tranziției T1 incrementează numărul de marcaje pentru ambele locații la infinit. .

4.5. Probleme suplimentare utilizând PIPE

1. Pentru început identificați opțiunile de editare a rețelelor Petri utilizând aplicația PIPE. După aceea, reprezentați o rețea Petri la alegere.

(a) Plasați marcaje în locațiile rețelei şi analizați dinamica acesteia comparând studiul teoretic cu rezultatele aplicației.

(b) Generați graful de acoperire. Analizați rezultatul.

2. Utilizând problemele propuse în secțiunea 4.4, şi comparând rezultatele obținute analitic cu cele generate de aplicație, verificați rezultatele obținute cu ajutorul aplicației PIPE.

5. Modele temporizate pentru DES

5.1. Prezentarea lucrării

Pentru modelarea temporizată a *sistemelor cu evenimente discrete (DES)* se pot utiliza atât automate temporizate cât şi rețele Petri temporizate După reluarea unor noțiuni de la curs referitoare la modelele temporizate, în secțiunea 5.3. sunt propuse probleme.

5.2. Automate şi rețele Petri temporizate

- *Un automat de stări temporizat* este un automat de stări care conține şi o *structură de temporizare* (structură de timp). Structura de timp este o mulțime alcătuită din secvențe, câte una pentru fiecare eveniment, definind *durata de viață* a evenimentului şi reprezentând intrare pentru modelul temporizat. Aceste informații sunt utilizate pentru a genera secvența de evenimente care determină parcurgerea automatul de stări.

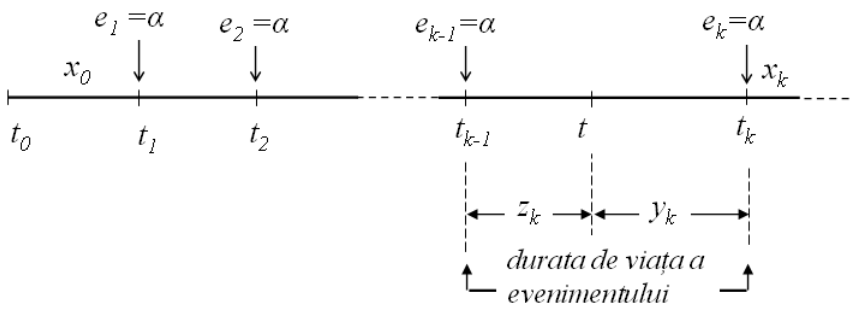


Figura 5.1. Diagramă de timp cu un eveniment permanent activ

Pentru diagrama de timp din figura 5.1 avem următoarele aspecte cunoscute:

E={α}, şi Γ(x)= {α} pentru toți x ∈ X,

evenimentele sunt $e_1, e_2, ... = \alpha$ şi au loc la momentele $t_1, t_2, \ldots$ şi se pot defini unele noțiuni.

Intervalul dintre 2 evenimente consecutive se numeşte *durata de viață* definit pentru figura 5.1 astfel $v_k=t_k - t_{k-1}$, $k=1, 2, \ldots$ cu $v_k \in R+$.

Evenimentul k este validat/activat la momentul t_{k-1}, fiind posibil în starea respectivă. La momentul t_k evenimentul are loc şi se definesc:

$y_k= t_k-t$ timpul evenimentului k, sau *timpul rezidual*,

$z_k= t-t_{k-1}$ vârsta evenimentului k.

Diagrama de timp este complet definită de înşiruirea tuturor duratelor de viață *{v1, v2, …}.*

- Un automat de stare temporizat se notează cu *(E, X, Γ, f, x_0, V),* unde

E este mulțimea de evenimente, *X* este spațiul stărilor (numărabile), x_0 starea inițială,

Γ(x) mulțimea evenimentelor 'posibile' din starea *x*, $x \in X$, $\Gamma \subseteq E$,

f funcția de tranziție $f: X \times E \rightarrow X$, definită în *X* doar pentru $e \in \Gamma(x)$,

$V = \{v_i: i \in E\}$ structura de timp (de temporizare).

- Într-un automat de stări temporizat, un eveniment *e* este *validat* dacă tocmai a avut loc şi rămâne posibil în starea următoare; sau dacă un alt eveniment are loc, cât timp *e* nu este posibil, şi în noua stare *e* este posibil. De câte ori un eveniment este activat, *timpului* lui i se atribuie valoarea corespunzătoare duratei de viață din structura de timp asociată.

- Un eveniment *e* este *invalidat/dezactivat* când un alt eveniment are loc şi determină tranziția spre o stare unde *e* nu este posibil. Dacă un eveniment este dezactivat atunci timpul asociat este ignorat.

- *Evenimentul declanşator* pentru o anumită stare este cel cu cea mai mică valoare a timpului dintre evenimentele validate.

- Pentru a analiza rețele Petri temporizate se introduce *secvența de temporizare* v_j asociată tranzițiilor t_j. Se utilizează un număr pozitiv real, $v_{j,k}$ asociat lui t_j care are următoarea semnificație: când tranziția t_j este activată de *k* ori, ea nu se declanşează imediat, prezentând o întârziere a declanşării dată de $v_{j,k}$; pe durata acestei întârzieri jetoanele sunt păstrate în locațiile de intrare ale tranziției t_j. Nu toate tranzițiile au întârzieri în declanşare (întârziere zero), aceste tranziții aparțin setului de *tranziții cu întârziere zero* $\in T_0$. Cele cu întârzieri sunt numite *tranziții temporizate* $\in T_D$, unde $T = T_0 \cup T_D$.

- În reprezentarea grafică, tranzițiile temporizate sunt notate cu ajutorul unor dreptunghiuri, spre deosebire de cele netemporizate notate cu o bară, iar durata de viață a tranziției este notată de obicei lângă dreptunghi.

- O rețea Petri temporizată este definită cu ajutorul setului (P, T, A, w, x, V):

P – mulțimea finită a stărilor, $p_i \in P$,

T – mulțimea finită a tranzițiilor, $t_i \in T = T_0 \cup T_D$,

A – mulțimea arcurilor, submulțime a mulțimii $(P \times T) \cup (T \times P)$,

w – funcția de pondere $w: A \rightarrow \{1, 2, 3, \ldots\}$,

x – marcarea inițială,

V – structura de timp a rețelei marcate (secvența de temporizare), $V \in \{v_j: t_j \in T_D\}$.

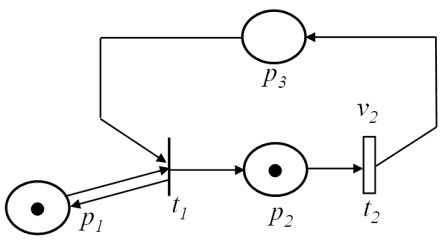


Figura 5.2. Rețea Petri temporizată. Exemplu

- În figura 5.2 se poate observa tranziția temporizată t_2 care are o întârziere de declanşare v_i. În acelaşi graf tranziția t_1 nu este temporizată.

5.3. Probleme propuse

1. Se analizează un sistem DES care are spațiul stărilor X= {x_1, x_2, x_3}, mulțimea evenimentelor E= {α, β, γ} şi diagrama tranzițiilor prezentată în figura 5.3. Se presupune că evenimentele α sunt planificate la momentele de timp (0., 1., 2.3, 5.6), evenimentele β sunt planificate la momentele de timp (0.2, 3.1, 6.4, 9.7) iar evenimentele γ sunt planificate la momentele de timp (2.2, 5.3).

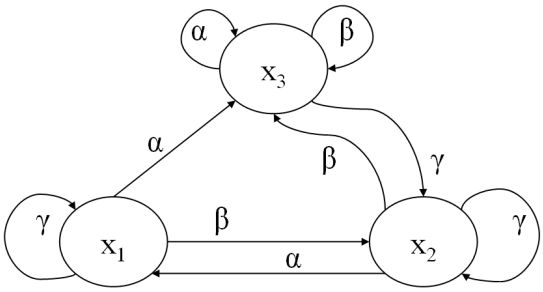


Figura 5.3. Diagrama tranzițiilor pentru un DES

(a) Presupunând că x_1 este starea inițială, să se construiască o diagramă de timp pentru secvența de evenimente din intervalul [0., 10.]. Se vor indica momentele în care apar evenimente, tipul lor şi starea sistemului pentru orice interval dintre 2 evenimente (nu trebuiesc reprezentate stările pentru toate momentele de timp).

(b) Încercați, pe baza diagramei de timp de la punctul (a), să estimați care este în general probabilitatea ca sistemul să fie în starea x_2.

2. Rețeaua Petri din figura 5.5 modelează execuția unui task T_3. simultan cu două task-uri secvențiale T_1 şi T_2 iar task-ul T_4 combină rezultatele de la T_3 şi T_2 după cum se poate observa în figura 5.4.

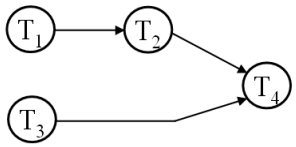


Figura 5.4. Sistem cu task-uri realizate serial şi în paralel

Presupunem că dorim să simplificăm modelul prin eliminarea tranziției t_{23} astfel încât tranzițiile t_2 şi t_3 să poată plasa un jeton direct în locația p_4 şi în plus atribuind greutatea 2 arcului (p_4, t_4).

(a) Este posibilă această modificare? Argumentați răspunsul;

(b) Pentru rețeaua Petri din figura 5.5 scrieți un set de ecuații recursive pentru momentele de declanşare a tranzițiilor.

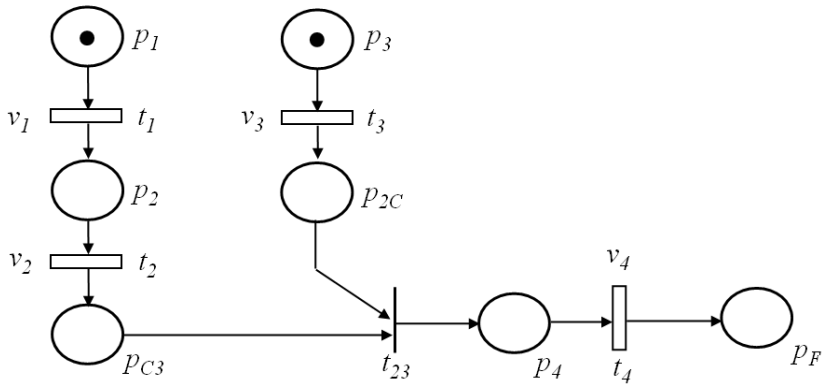


Figura 5.5. Rețea Petri temporizată

(c) Să se repete problema (b) pentru modelul modificat. Care este diferența în acest caz?

6. Modelarea unor procese economice utilizând lanțuri Markov

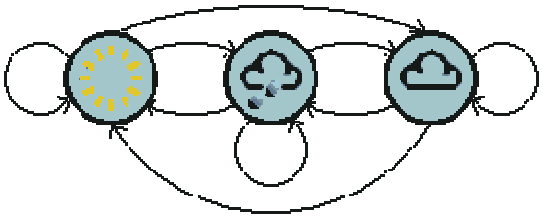
6.1. Analiza proceselor stohastice utilizând lanțuri Markov

Această lucrare adresează aplicațiile modelării şi simulării. În lucrare se analizează un exemplu de utilizare a Lanțuri Markov în domeniul prognozei de marketing. Lanțurile Markov sunt modele care permit modelarea stohastică. Chiar dacă se foloseşte un model simplificat, scopul este să subliniem utilitatea modelării.

În această lucrare utilizăm doar lanțuri Markov în timp continuu – *DTMC (Discrete Time Markov Chain)* pentru studiul de caz.

Prin lanț Markov se înțelege o secvență de variabile aleatoare X_1, X_2, X_3, ... care au proprietatea lui Markov. Proprietatea lui Markov poate fi exprimată în mai multe moduri: „data fiind starea prezenta, starea viitoare şi starea trecută sunt independente" sau „starea următoare este determinată de starea curentă".

Toate valorile X_i posibile formează mulțimea stărilor S, denumită spațiul stărilor. De exemplu, simplificat, starea vremii poate fi: însorit, (cer acoperit) noros, ploaie. În figurile 6.1 este reprezentat graful orientat, tabelul de corespondență şi starea inițială.



vremea ieri		*vremea azi*		
		insorit	**noros**	**ploios**
	insorit	0.5	0.25	0.25
	noros	0.375	0.125	0.375
	ploios	0.125	0.625	0.375

insorit	**noros**	**ploaie**
1.0	0.0	0.0

Figura 6.1. Modelare simplificată pentru prognoză meteo. Matricea tranzițiilor şi vectorul inițial.

Lanţurile Markov sunt descrise cu ajutorul graf-urilor orientate. Pe arce pot fi notate probabilităţile de tranziţie.

Probabilităţile de tranziţie sunt descrise cu o *matrice a tranziţiilor*. În această matrice a tranziţiilor, suma intrărilor pe coloane/rânduri =1.

Se mai defineşte *vectorul iniţial* sau *distribuţia iniţială* care este starea la momentul 0, în cazul figurii 6.1 este vectorul de jos.

Aşa cum specifică numele, DTMC (Discrete Time Markov Chain) consideră intervale de timp discret. (ex. la intervale de timp Δt).

DTMC sunt caracterizate cu ajutorul unei matrice a tranziţiilor pentru un pas, diagrama tranziţiilor (matricea P, dimensiuni NxN cu elementele p_{ij} – reprezintă probabilitatea tranziţiei din starea *i* in starea *j*, intr-un pas). După *n* paşi, prin multiplicarea matricei cu ea însăşi, când *n* tinde la infinit obţinem *matricea staţionară*. În cazul figurii 6.1 matricea P este dată la mijlocul figurii.

Evident, exemplul din figura 6.1 ignoră alţi factori climatici care influenţează prognoza vremii (curenţii de aer, etc.).

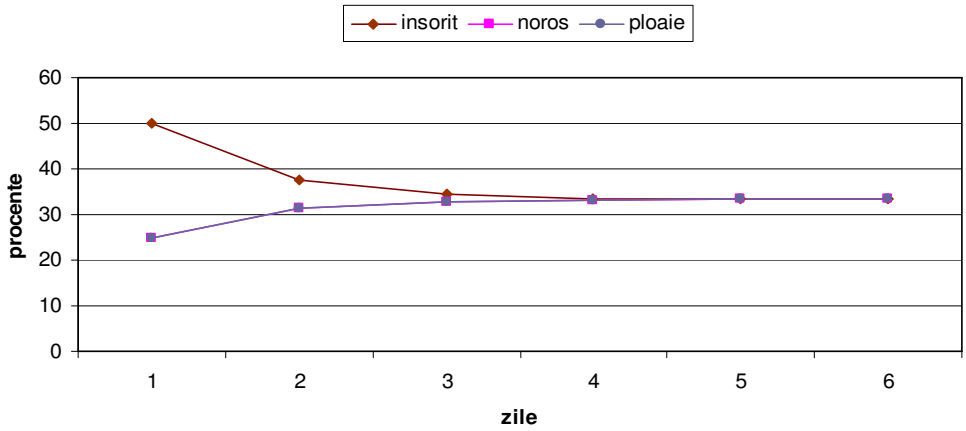


Figura 6.2. Evoluţia după 5 paşi a exemplului din figura 6.1.

Figura 6.2 prezintă evoluţia rezultatele procesului de evaluare succesivă a prognozei pentru exemplul din figura 6.1. Vectorul iniţial a fost înmulţit cu matricea tranziţiilor pentru a obţine evaluarea pentru prima zi, şi apoi rezultatul obţinut fiind din nou înmulţit cu matricea tranziţiilor pentru a obţine evaluarea pentru ziua a doua, etc. Distribuţia staţionară are toate cele trei probabilităţi 33.33%.

6.2. DTMC – Studiu de caz. Evoluţia ponderii de piaţa[1]

În această secţiune este prezentat un studiu de caz pentru o analiză a evoluţiei ponderii de piaţă a unor produse concurenţiale.

[1] C. Raţiu-Suciu, F. Luban, D. Hîncu, M. Sarchiz, Modelarea si simularea proceselor economice, Ed. Didactică şi pedagogică, Bucuresti, 1997

În acest studiu de caz se presupune că avem trei produse concurente de pe piaţa produselor cosmetice pentru copii. Cele trei produse le vom identifica cu C1, C2 respectiv C3. Cota de piaţă a fiecărui produs este 35% pentru produsul C1, 45% pentru produsul C2, şi respectiv 20% pentru produsul C3.

Coeficientul de fidelitate de la o lună la alta este constant şi anume: 55% dintre cumpărătorii actuali ai produsului C1 rămân fideli acestui produs, 65% rămân fideli produsului C2 respectiv 75% rămân fideli produsului C3. Ceilalţi cumpărători se reorientează spre celelalte produse disponibile, după cum se poate observa în tabelul 6.1.

Tabelul 6.1. Reorientări ale clienţilor

Produs 'părăsit'	Reorientări (%)		
	C1	C2	C3
C1	-	20	25
C2	15	-	20
C3	10	15	-

Considerând constante probabilităţile de tranziţie se analizează evoluţia ponderii de piaţă a celor trei produse pentru şase luni.

Pe baza acestui studiu se pot face evaluări ale evoluţiei şi astfel pot fi stabilite politici manageriale potrivite pentru fiecare produs în parte la firma la care este produs/distribuit.

Astfel, se ştiu:

Ponderea pe piaţa a 3 produse (la un moment dat, iniţial – adică cota de piaţă iniţială),

Rezultatele unui sondaj de piaţă (din care să rezulte opţiunea cumpărătorilor, pentru următoarele perioade de timp – adică matricea de reorientare a clienţilor).

Modelare cu lanţuri Markov utilizează matricea stohastică (matricea de reorientare a clienţilor în care s-au inserat probabilităţile celor fideli) şi distribuţia iniţială (adică cota de piaţă iniţială).

Etapele sunt următoarele:

1. se construieşte matricea probabilităţilor pe baza coeficienţilor de fidelitate şi a tabelului de reorientare. În această matrice, toate valorile sunt pozitive şi suma lor e 1 (100% dacă utilizăm procent) pentru fiecare linie.

$$P = \begin{pmatrix} 0.55 & 0.20 & 0.25 \\ 0.15 & 0.65 & 0.20 \\ 0.10 & 0.15 & 0.75 \end{pmatrix}$$

2. se scrie distribuţia iniţială sub forma unui vectorul linie, format din cota de piaţă iniţială.

$$a = \begin{pmatrix} 0.35 & 0.45 & 0.20 \end{pmatrix}$$

3. se determină cota de piaţă după prima perioadă înmulţind distribuţia iniţială cu matricea probabilităţilor. Acest rezultat va înlocui distribuţia iniţială în calculul cotei de piaţă pentru al doilea interval de timp. Se continuă pentru tot intervalul analizat.

Astfel pentru acest exemplu avem, după o lună:

$$a \times P = (0.28 \quad 0.3925 \quad 0.3275)$$

Iar după două luni:

$$a \times P \times P = (0.2455 \quad 0.3602 \quad 0.3941)$$

În final se obține:

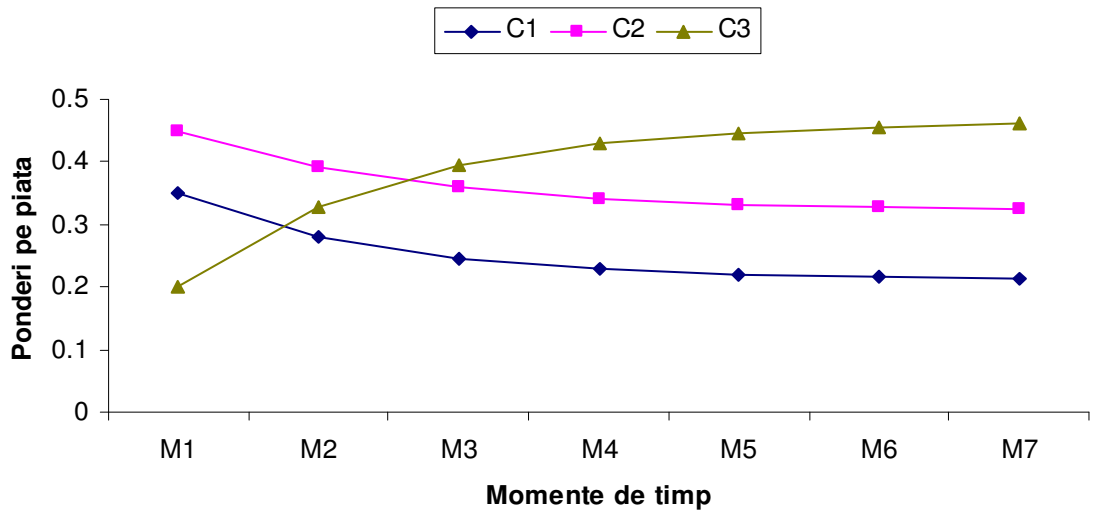


Figura 6.3. Evoluția pentru 6 luni a cotei de piață utilizând lanțuri Markov

Pe baza acestor cote se poate decide reducerea de preț pentru produsele C1 şi C2 pentru a creşte vânzările sau, dacă nu se poate, se renunță la desfacerea lor.

6.3. Studii propuse

1. Utilizând etapele descrise în secțiunea anterioară, să se facă o analiză a evoluției pe piață a trei produse ştiind următoarele:

- produsul P1, concurent cu P2, şi P3, produse similare
- după un studiu de marketing au rezultat următoarele:
  - firma deține o pondere de 44,9% din piață,
    - comparativ cu 25,1% şi respectiv 30% concurența
  - există o relativă stabilitate a preferințelor consumatorilor
    - 80% din cei ce au achiziționat P1 vor continua să-l achiziționeze, pt. P2 şi P3 proporția cumpărărilor fideli fiind de 50% respectiv 75%
    - reorientările sunt în tabelul 6.2.

Presupunând că nu apar modificări pe piață, se dorește cunoaşterea evoluției ponderii de piață a celor 3 produse pentru următoarele 6 momente de timp. De asemenea se presupune constant numărul de produse şi numărul de consumatori. Propuneți soluții pentru producători.

Tabelul 6.2. Reorientări ale clienților

Produs 'părăsit'	Reorientări (%)		
	P1	P2	P3
P1	-	10 %	10 %
P2	35 %	-	15 %
P3	10 %	15 %	-

2. Presupunem că pe o piață sunt 3 tipuri de calculatoare PC1, PC2, PC3. Se ştiu următoarele:

- cota de participare în totalul pieței 51%, 39% şi 10%
- din sondajele efectuate au rezultat clienți fideli
  - 65% pentru PC1, 75% pentru PC2, şi 85% pentru PC3
  - ceilalți clienți se reorientează conform tabelului 6.3.

Tabelul 6.3. Reorientări ale clienților pentru problema 2.

Produs 'părăsit'	Reorientări (%)		
	P1	P2	P3
P1	-	15	20
P2	15	-	10
P3	5	10	-

Se cere să analizați evoluția ponderii pe piață a produselor pentru minim 3 luni presupunând constante probabilitățile de tranziție. Care ar fi propunerile d-voastră pentru a îmbunătăți situația celor care pierd din ponderea de piață?

Bibliografie

Discrete Event Systems: Modeling and Performance Analysis, Christos G. Cassandras, Richard D Irwin, 1993.

Modelarea şi simularea proceselor economice, M. Stoica, I. Ionita, M. Botezatu, Ed. Economica, Bucureşti, 1997.

Modelarea şi simularea proceselor economice, C. Rațiu-Suciu, F. Luban, D. Hîncu, M. Sarchiz, Ed. Didactică şi pedagogică, Bucuresti, 1997.

Aplicații ale rețelelor Petri în studierea sistemelor cu evenimente discrete, Octavian Păstrăvanu, Mihaela Matcovschi Cristian Mahulea, Editura Gh. ASACHI, 2002.

Introduction to Computational Science: Modeling and Simulation for the Sciences, Angela B. Shiflet & George W. Shiflet, Princeton University Press, 2006.

Introduction to Discrete Event Systems, second edition, Christos G. Cassandras & Stephane Lafortune, Springer, 2008.