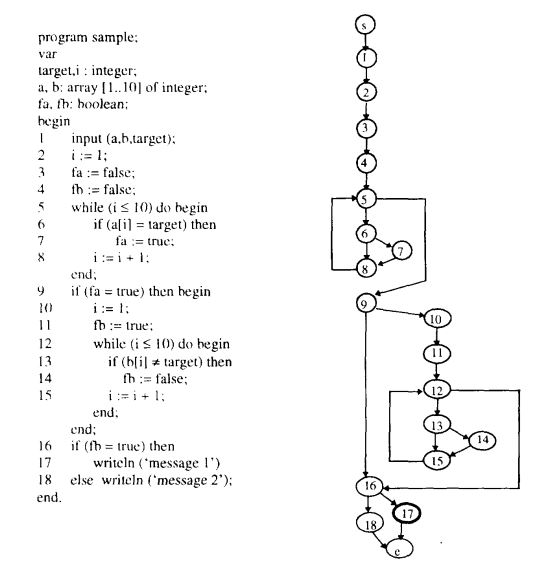
Teoria Sistemelor – The Chaining Approach

Sectiunea 2: **Concepte de Baza** – Graful: Componente. Definitie

Vom reprezenta structura unui program folosind un **graf de control al fluxului de date**. Un astfel de graf C = (N, A, s, e); unde N = setul de noduri, A = relatia binara pe N/ setul de muchii si s & e reprezinta date de I/O unice, din multimea N.​

Daca n din N este **nod de test** (adica are o conditie - if sau este un loop – for/while/do…while), atunci muchia derivata din el poarta numele de **branch**.​ Daca un branch prezinta o conditie pentru o metoda de decizie sau una repetitive, va devein predicatul branch-ului.

Branch Predicate

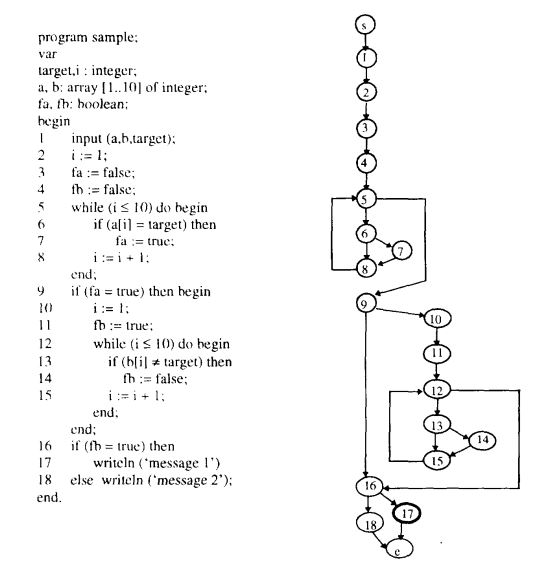


Data de intrare se refera la inputul programului si reprezinta un vector de variabile x(1), x(2), .., x(n), fiecare x(i) avand cate un domeniu de definitie D(x(i)), care, prin produsul cartezian al tuturor domeniilor variabilelor, formeaza domeniul D al programului.​

Patch-ul sau ruta sugereaza secventa (=distanta parcursa) dintre 2 noduri, care poate fi realizabila daca executia nodului se realizeaza cu succes sau nerealizabila, altfel.​

Variabila are 3 stari: de asignare, de iesire(output), de predicat al unei conditii/loop.​

Sectiunea 4: **Concepte Generale** – The Chaining Approach

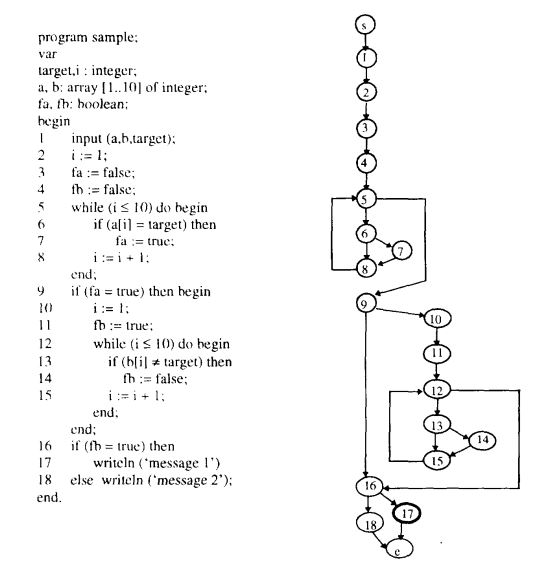
Asa cum am specificat, avem 3 metode de testare automata pentru generarea datelor: random, path-oriented, goal-oriented. Studiul nostru, The Chaining Approach, reprezinta o extindere a metodei goal-oriented, in sensul ca goal-oriented este limitat in procesul de cautare, unele noduri ramanand neacoperite.

Cum functioneaza metoda studiata de noi?

Se efectueaza testarea executiei unui program arbitrar cu input x. Pentru fiecare branch se incepe procesul de cautare (care vom vedea ca este pe nivele de cautare) astfel: se face executarea pe acelasi branch sau o va lua pe alt branch pentru a ajunge la nodul care reprezinta scopul executiei.

Ce este diferit la metoda studiata de noi este faptul ca aceasta foloseste **conceptul de raport de dependenta de control** (=last definition). In acest sens, se studiaza relatia de dependenta dintre nodul/nodurile de test si nodul/nodurile alese sa execute aceste noduri de test.

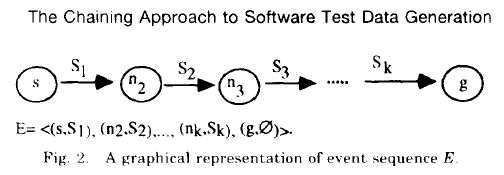
Last Definition(16): se atribuie valoare variabilei

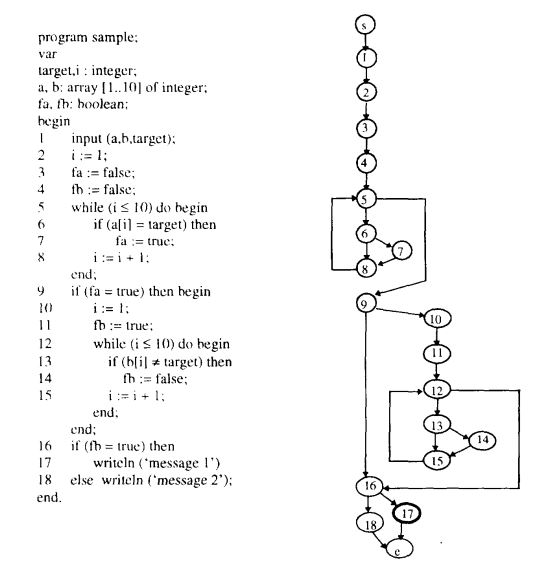


Last Definition(16): se schimba valoarea variabilei din cauza predicatului de decizie

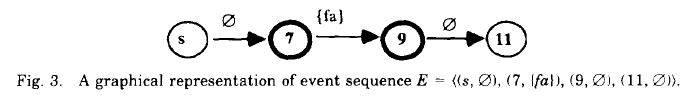
Last Definition(16): nodul definit de variabila noastra

**Conceptul de secventa de evenimente**: este generat de metoda studiata de noi, abordarea inlantuita si apoi, folosit pentru a ghida executia programului in timpul procesului de cautare. Evenimentul este echivalent executiei unui nod. Dezvoltand acest lucru, afirmam faptul ca o secventa E de evenimente reprezinta sirul de evenimente e(1), e(2), …, e(n) unde fiecare e(i) este de forma e(i) = (n(i), S(i)) unde n(i) reprezinta un nod din multimea de noduri N si S(i), un set de variabile definit ca set de constrangeri (=conditia impusa la executia a 2 noduri). Acest set de variabile nu trebuie sa se modifice in timpul executiei programului.



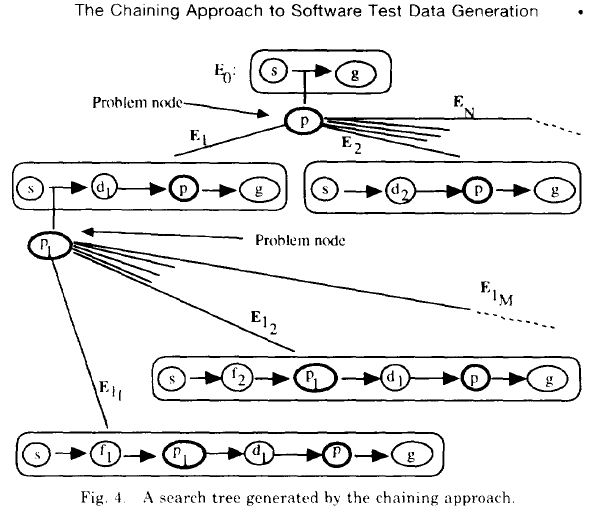


Sa presupunem ca nodul nostru tinta este 11. Deci secventa va incepe cu executia nodului s si se va termina cu executia nodului 11. Vom alege arbitrar un input x: a = (27, 58, 78, 4, 89, 21, 54, 85, 35, 96), b = (45, 99, 6, 45, 2, 63, 28, 15, 94, 22), target = 56. Observam ca branch-ul deciziei false pentru (9,16) nu conduce spre nodul 11, deci aici executia se va suspenda. In acest moment, procesul de cautare incearca gasirea unui nou input pentru a schimba ramura deciziei de la 9. Cum acest lucru este imposibil chaining approach identifica nodul 9 ca fiind nodul problema si ii cauta un set de LD, adica un set de noduri care asigneaza o valoare variabilei (={fa}) care poate fi folosita de nodul 9. Si astfel LD(9) = {3, 7}. Deoarece nodul 3 a fost deja executat inaintea lui 9, vom alege nodul 7(executarea lui e limitata de conditie). Astfel avem executarea lui 7 prima data, urmata de executarea lui 9, apoi, in final, executarea lui 11. Ruta s-7 este traversata oricum, apoi pentru 7-9 vor fi folosite rute care nu vor modifica valoarea variabilei {fa} inainte de nodul 9. Si astfel, a fost generata o secventa.

**E = {(s, Ø), (7, {fa}), (9, Ø), (11, Ø)}** cu 4 evenimente **e(1)=(s, Ø), e(2)= (7, {fa}), e(3)= (9, Ø), e(4)= (11, Ø)**. Secventa data presupune ca nodul s sa fie executat primul, urmat de executia nodului 7, urmat de executia lui 9 si in final, a lui 11. O constrangere este impusa la executia dintre nodurile 7 si 9 (= {fa}) si este necesar faptul ca valoarea acestei variabile sa nu se modifice in timpul executiei programului intre nodurile 7 si 9. “Ø” reprezinta faptul ca nu avem constrangeri impuse la executia dintre s-7 si 9-11, asta insemnand ca programul poate urma orice ruta gaseste intre s-7 si 9-11.

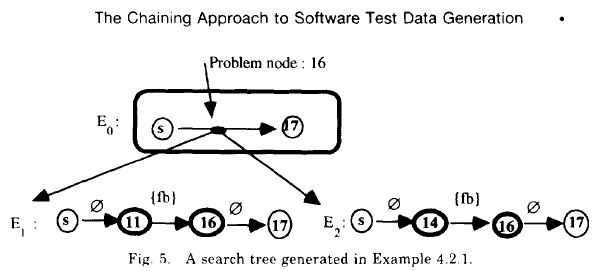
O astfel de secventa este **realizabila** daca exista un input care sa traverseze secventa respectiva - pentru programul nostru avem: a = (56, 58, 78, 4, 89, 21, 54, 85, 35, 96), b = (45, 99, 6, 45, 2, 63, 28, 15, 94, 22), target = 56. Secventa **E’ = {(s, Ø), (3, {fa}), (9, Ø), (11, Ø)}** nu este **realizabila**, deoarece nu avem input.

Care este mecanismul de generare de secvente E?

Pentru un nod scop “g”, adica nodul la care dorim sa ajungem, se va genera o secventa initiala de evenimente **E(0) = {(s, Ø), (g, Ø)}**. Presupunem ca in timpul procesului de cautare, apare un nod de test “p”, datorita caruia executia ar trebui sa se schimbe, dar procesul meu de cautare (va fi explicat in cele ce urmeaza) nu poate gasi un input capabil sa modifice executia programului in nodul de test. In acest scenariu, chaining approach gaseste un set de LD(p) de forma {d(1), d(2), …, d(n)}, aceste set generand n secvente, de forma: **E(i)= {(s, Ø), (d(i), D(d(i))), (p, Ø), (g, Ø)}**; s = nodul de start, d(i) = LD pentru p, D(i) = domeniul lui d(i), p = nodul care schimba executia, g = nodul tinta/destinatie. Daca se intampla sa mai apara un nod problema intre s si p, se va creste nivelul de cautare, adica va aparea un p(1), care va avea un set de LD(p(1)) generat la fel ca in primul caz si secventele vor deveni: **E(i)= {(s, Ø), (f(i), D(f(i))), (p(1), Ø), (d(i), D(d(i))), (p, Ø), (g, Ø)}**. Se va organiza procesul de cautare sub forma unui arbore de cautare.

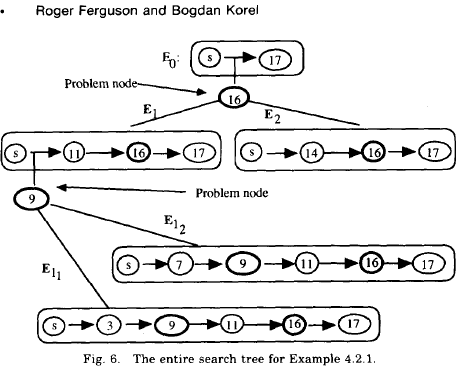
Radacina

Copii

Sa presupunem ca avem nodul 17 ca nod destinatie pentru programul nostru. Se va genera astfel o secventa initiala **E(0) = {(s, Ø), (17, Ø)}** si incepe cautarea executand programul pentru un input arbitrar x: a = (27, 58, 78, 4, 89, 21, 54, 85, 35, 96), b = (45, 99, 6, 45, 2, 63, 28, 15, 94, 22), target = 56. Pentru acest input, observam ca programul are un nod 16 ca nod problema, intrucat acolo este un factor de decizie care ii schimba cursul, nodul 17 neputand fi atins.

Se cauta LD(16). Se gasesc nodurile 4, 11, 14 (={fb}). Se vor genera **E(1)= {(s, Ø), (11, {fb}), (16, Ø), (17, Ø)}** si **E(2)= {(s, Ø), (14, {fb}), (16, Ø), (17, Ø)}**. Se va lucre intai cu E(1). Se gaseste un nod problema la 9. Se va genera **E(1.1) = {(s, Ø), (3, {fa}), (9, Ø), (11, {fb}), (16, Ø), (17, Ø)}** si **E (1.2)= {(s, Ø), (7, {fa}), (9, Ø), (11, {fb}), (16, Ø), (17, Ø)}**. Si in urma prelucrarii, ramane E(1.2) si inputul: a =(56, 58, 78, 4, 89, 21, 54, 85, 35, 96), b = (56, 56, 56, 56, 56, 56, 56, 56, 56,

56), target = 56.

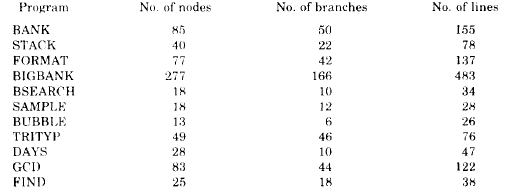


Sectiunea 6: **Studiul Experimental** – The Chaining Approach

Scopul experimentului a fost testarea generarii de date random, testarea generarii de date goal-oriented (~chaining approach cu nivel 0 de cautare), testarea generarii de date path-oriented si testarea chaining approach pe nivelele 1 si 3 de cautare.

Programe Pascal. Numar: 11. Tip:

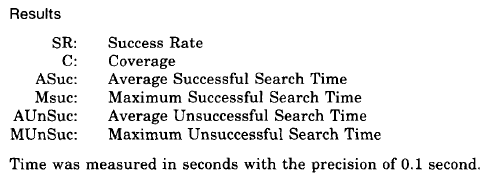
1. Bank – mentinerea unui cod bancar si realizarea unor tranzactii simple
2. Stack – realizarea operatiilor stack: push, pop, empty, full
3. Format – formatare de text
4. BigBank – versiune extinsa a lui Bank; suporta tranzactii pentru diferite tipuri de conturi bancare
5. BSearch – program de cautare binara
6. Sample – folosit in exemplul nostru (fig1)
7. Bubble – sortare de array-uri folosind metoda bubble sort
8. Trityp – caracteristicile unui triunghi si categorizarea lui in echilateral, isoscel, dreptunghic
9. Days – calculeaza nr de zile dintre 2 date calendaristice
10. GCD – CMMDC a unei secvente de numere
11. Find – are ca input un array A si un index F si intoarce un array format din elementele care sunt <= F sau >= F



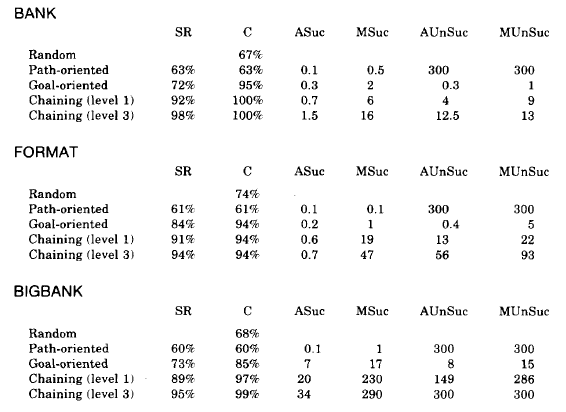
Experimentul a constat in implementarea celor 3 metode: goal-/path-oriented si chaining-approach. S-au folosit 11 programe, iar in fiecare program, scopul rularii lui a fost ca pentru fiecare nod din el sa se gaseasca un input care sa demonstreze ca nodul selectat a fost executat. O parte din datele initiale au fost generate random, dar unele noduri depind de data de intrare, asa ca pentru a se asigura acuratetea inputurilor, cautarea a fost executata de 10 ori. Astfel a fost facut un numar de (10\*nr\_noduri) incercari pentru fiecare program. A fost setata, de asemenea, o limita pentru executia incercarii, 5 minute, care daca se depasea, se socotea ca fiind incercare esuata. Pentru abordarea random, am avut inputurile alese random in limita a 5 minute si daca cel putin un input traversa nodul si acesta era executat, cautarea era realizata cu success si inputul, retinut. Pentru path-oriented, s-au generat toate rutele aciclice ca sa se ajunga la nodul selectat. Daca nu se gaseau rute, atunci se generau cele corespunzatoare metodelor repetitive, cu o singura iteratie. Daca nici asa nu functiona, se trecea la generarea celor cu 2 iteratii si tot asa, pana se gasea un rezultat sau se scurgea timpul.

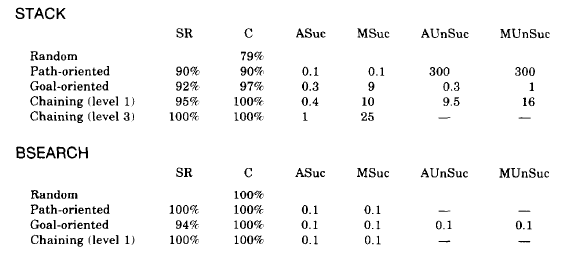
In timpul acestui studiu, s-a urmarit:

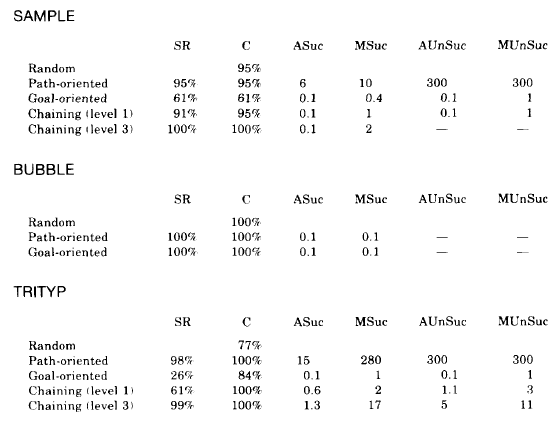
1. Rata de succes
2. Acoperirea (= % de noduri pentru care cel putin o incercare a fost realizata cu succes, in gasirea input-urilor)
3. Timpul mediu al cautarilor efectuate cu succes
4. Timpul maxim al cautarilor efectuate cu succes
5. Timpul mediu al cautarilor efectuate fara succes
6. Timpul maxim al cautarilor efectuate fara succes

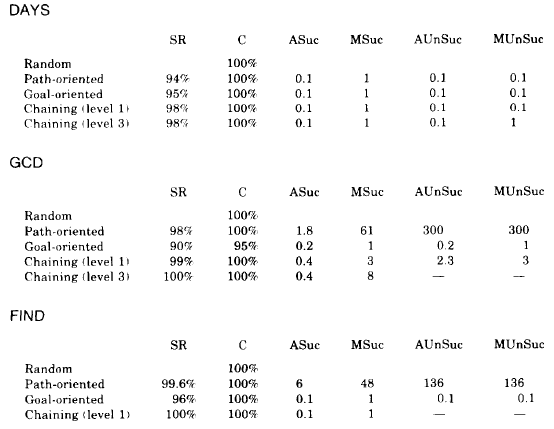


Rata de succes este definita sub forma: SR = ; unde m = 10(factorul de repetitie, adica a fost repetat de 10 ori pentru fiecare nod).









Ca si concluzie a experimentului, putem afirma faptul ca metodele testate se comporta diferit in functie de complexitatea programului. In acest sens, vorbim despre programe cu o structura simpla, carora metodele random si path-oriented le-au facut fata fara probleme. In schimb, pentru programele complexe a rezultat faptul ca chaining approach este o solutie optima in vederea generarii datelor.