Acest document conține demonstrația teoremei lui Cook, cu unele adăugiri în raport cu cartea "Introducere în Analiza Algoritmilor". Util este, mai ales, exemplul de construcție a formulei $\mathbf{F_{Alg,D}}$ pentru un algoritm nedeterminist simplu.

3.2.4 Teorema lui Cook

Cunoaşterea unei probleme **NP-complete** este deosebit de importantă, fiind baza construcției întregii ierarhii **NPC**. O asemenea problemă este cea a satisfiabilității unei formule în calculul propozițional, iar **NP-**completitudinea ei este consecința teoremei lui Cook.

Definiția 3.19 Numim formulă propozițională în formă normală conjunctivă (pe scurt **CNF**) o formulă cu sintaxa convențională:

<variabilă> ::= un simbol care poate fi asociat unei unice valori de adevăr din mulțimea **{0,1}**
<literal> ::= <variabilă> | ¬<variabilă>
<termen> ::= <literal> | (<literal> [∨ <literal>]$^{+}$)
<formulă> ::= <termen> [∧ <termen>]*

unde ¬ este negația logică, iar ∨ şi ∧ sunt conectorii logici **sau**, **şi**. Considerăm că un termen nu are apariții redundante ale literalilor (o variabilă poată să apară doar de două ori într-un termen, cu şi fără negație), iar formula nu conține termeni identici.

O formulă **CNF F** este satisfiabilă dacă există o asociere (legare) a variabilelor din **F** la valori de adevăr astfel încât valoarea lui **F** să fie **1**.

Fie problema: pentru o formulă **CNF F** să se determine dacă **F** este satisfiabilă. Problema poartă numele de problema **SAT**: problema satisfiabilității unei formule propoziționale în formă normal conjunctivă. De exemplu, formula **F = (x_1 ∨ ¬x_2 ∨ x_3) ∧ (x_1 ∨ x_2 ∨ ¬x_3)** este satisfăcută pentru **x_1=1**, indiferent de valorile celorlalte variabile.

Teorema 3.7 (Cook) **SAT ∈P** dacă şi numai dacă **P = NP**.

P = NP ⇒ SAT ∈P. Să considerăm următorul algoritm nedeterminist de rezolvare a problemei **SAT**:

```
N_SAT(F) {
   // F este formula CNF
   Var = variabile(F); // card(Var) = n
   for-each(x∈Var) x = choice({0,1});

   // Verificare satisfacere F în raport cu valorile variabilelor
   for-each(term ∈ termeni(F)) {
       satisf_term = 0;
       for-each(x∈Var)
          if((literal x)∈term ∧ x=1 ∨ (literal ¬x)∈term ∧ x=0)
            {satisf_term = 1; break;}
       if(¬satisf_term) fail;
   }
   success;
}
```

Dacă **n** este numărul variabilelor din formulă, iar $m=O(n^k)$ este numărul termenilor, limitat polinomial în raport cu **n**, atunci complexitatea algoritmului **N_SAT** este polinomială: $O(n^{k+c+1})$, unde $O(n^c)$ este complexitatea căutării unui literal într-un termen (operațiile **(literal x)∈term** şi **(literal ¬x∈term)**). Prin urmare **SAT∈NP**. Dacă **P = NP** şi **SAT ∈NP** rezultă imediat **SAT ∈P**.

SAT∈P ⇒ P = NP. Implicația se demonstrează arătând că pentru orice algoritm **Alg∈NP** - cu complexitatea polinomială **p(n)**, unde **n** este dimensiunea datelor **D** ale lui **Alg** - există o formulă **CNF** echivalentă, fie ea $F_{Alg,D}$, astfel încât:

- $F_{Alg,D}$ are un număr polinomial de variabile şi termeni în raport cu **p(n)** şi, de asemenea, se poate construi în timp polinomial în raport cu **p(n)**.

- $F_{Alg,D}$ este satisfiabilă dacă şi numai dacă **Alg** se termină cu succes pentru datele de intrare **D**.

Cu alte cuvinte, se arată că pentru orice problemă **Q ∈ NP** avem $Q \leq_p SAT$. Deoarece **SAT∈ NP**, rezultă că **SAT** este **NP-completă** şi, conform teoremei (3.5), dacă **SAT ∈P**, atunci **P = NP**.

Construcția mai uşoară a formulei $F_{Alg,D}$ impune restricționarea modului de specificare a algoritmului **Alg**, dar fără a afecta funcționalitatea acestuia. Restricțiile se referă la structura de control şi la variabilele algoritmului.

Controlul execuției algoritmului

- Instrucțiunile algoritmului sunt etichetate **1,2,...,l**, în secvență de la prima la ultima instrucțiune din specificația textuală a algoritmului. Prima instrucțiune executată este instrucțiunea cu eticheta **1**.

- Controlul execuției se realizează folosind doar două instrucțiuni de salt: **if(variabilă) goto i** şi, respectiv, **goto i**, pentru o etichetă **i∈[1,l]**. Predicatele sunt simple variabile cu valori booleene. Valorile expresiilor mai complicate trebuie atribuite unor variabile pentru a fi testate.

Variabilele algoritmului

- În algoritm nu apar constante. Constantele sunt reprezentate prin variabile considerate ca parametri suplimentari ai algoritmului.

- Inițial, valoarea oricărei variabile a algoritmului este **0**. Excepție fac doar variabilele cu rol de parametru, inițializate cu datele **D** ale algoritmului sau cu valorile constante reprezentate de unele variabile (valori considerate parte a datelor **D**).

- Considerăm că valoarea oricărei variabile este reprezentată prin **w** biți, numerotați **1..w** de la dreapta spre stânga. Prin convenție, o valoare cu bitul de rang **1** egal cu **1** corespunde valorii **adevărat**, iar o valoare cu bitul de rang **1** egal cu **0** corespunde valorii **fals**.

- Variabilele din algoritmul **Alg** sunt scalare, vectorii fiind reprezentați prin mulțimi de variabile scalare. De exemplu, vectorul **v** cu **m** elemente este reprezentat de variabilele **v1, v2,...,vm**, iar atribuirea **x = v_i** este înlocuită prin codul de mai jos, unde variabila **val_k**, **k=1,2,...,m**, reprezintă constanta **k**, iar variabila **val_v** reprezintă valoarea variabilei **v_i**.

```
e1: test = i≠val_1;
    if(test) goto e2;
    val_v = v1;
    goto em+1;
    ...
ek: test = i≠val_k;
    if(test) goto ek+1;
    val_v = vk;
    goto em+1;
    ...
em+1: x = val_v;
```

} cod indexare v_i

- Deoarece se consideră că fiecare operație efectuată în cursul execuției algoritmului **Alg** durează o unitate convențională de timp, numărul variabilelor folosite de algoritm nu poate depăşi **c p(n)**, cu **c** constant. Notăm cu **Var** mulțimea variabilelor folosite de algoritm, **card(Var)≤c p(n)**.

Restricțiile introduse nu reduc puterea computațională a algoritmilor. Este suficient să ne gândim la un algoritm (determinist) direct codificabil într-un limbaj de asamblare sau la codul nativ generat de un compilator.

Construcția formulei **$F_{Alg,D}$** priveşte o cale posibilă din execuția algoritmului nedeterminist **Alg** ca pe o secvență de stări. O stare este un triplet **s = ⟨e,var,t⟩**, unde **t** este momentul de timp corespunzător stării **s**, **e** este eticheta instrucțiunii executată la momentul **t**, iar **var** este mulțimea valorilor variabilelor **Var** ale algoritmului la momentul **t**. Timpul curge începând de la **1** (momentul inițial, cel al execuției instrucțiunii **1**) cu increment unitar până la momentul **p(n)**, când execuția se termină. Formula **CNF $F_{Alg,D}$** trebuie construită în aşa fel încât:

1. Să reflecte condițiile de corectitudine ale oricărei secvențe de stări **⟨e_1,var_1,1⟩,...,⟨e_k,var_k,p(n)⟩** ce poate rezulta în urma execuției algoritmului nedeterminist **Alg**.

2. Să fie satisfiabilă dacă şi numai dacă există o secvență de stări care corespunde execuției cu succes (execuție serială) a unei căi din arborele copiilor algoritmului.

Formula nu va descrie fiecare flux particular de instrucțiuni executate de algoritm, ci doar condițiile ca un flux corect (oricare flux, adică o secvență corectă de stări) să existe. Interesantă este, din acest punct de vedere, şi modalitatea reprezentării atribuirii **v = choice({S_1, S_2,...,S_k})** şi, implicit, descrierea funcționării algoritmului fără a-i reprezenta explicit cele **k** copii. Efectul atribuirii este descris ca o disjuncție de termeni din formula **$F_{Alg,D}$**, termeni ce desemnează toate

valorile posibile ale variabilei **v** la un moment dat de timp. În acest mod, putem descrie un flux generic de instrucțiuni ale lui **Alg** fără a preciza cărei copii a algoritmului îi aparține fluxul. Cu alte cuvinte, putem descrie simultan toate fluxurile corecte de instrucțiuni pe care le poate genera algoritmul în cursul execuției sale.

Construcția formulei $F_{Alg,D}$ utilizează următoarele clase de variabile[1] cu valori de adevăr **{0,1}**.

1. Variabile de forma **B(x,i,t)**, **x∈Var**, **i∈1..w**, **t∈1..p(n)**. **B(x,i,t)** corespunde bitului de rang **i** al valorii variabilei **x** la momentul **t**. **B(x,i,t)=1** arată că bitul **i** este **1** la momentul **t**, iar **B(x,i,t)=0** arată că bitul **i** este **0** la momentul **t**. Astfel, formula $\bigwedge_{i=1,w} \neg B(x,i,t)$ este satisfiabilă dacă şi numai dacă variabila **x** a algoritmului are valoarea **0** la momentul **t**.

2. Variabile de forma **S(k,t)**. **S(k,t)=1** indică execuția instrucțiunii **k** la momentul **t**, iar **S(k,t)=0** arată că la momentul **t** nu se execută instrucțiunea **k**.

Formula $F_{Alg,D}$ conține şase tipuri de sub-formule **CNF** care descriu corectitudinea stărilor pe care le poate genera orice flux al execuției instrucțiunilor algoritmului la diverse momente de timp $1 \le t \le p(n)$:

$$F_{Alg,D} = Init \wedge Start \wedge Serial \wedge Flow \wedge Eval \wedge Stop$$

Init descrie starea inițială a variabilelor algoritmului. La momentul **t = 1** variabilele din **Var** au valoarea **0**, cu excepția celor cu rol de parametru, care sunt inițializate conform datelor **D**.

$$Init = \bigwedge_{\substack{x \in Var \\ i=1,w}} T_{x,i}$$

$T_{x,i}$ = **B(x,i,1)**, dacă **x** este parametru al **Alg** şi bitul **i** din valoarea variabilei **x** este **1** conform datelor **D**

$T_{x,i}$ = ¬**B(x,i,1)**, dacă **x** nu este parametru al **Alg** sau dacă **x** este parametru şi bitul **i** din valoarea sa este **0** conform **D**

Se observă imediat că **Init** este **CNF** şi este satisfiabilă dacă şi numai dacă inițializarea variabilelor algoritmului este corectă.

Start arată că prima instrucțiune executată de algoritm este cea etichetată **1**.

$$Start = S(1,1) \wedge \bigwedge_{i=2,l} \neg S(i,1)$$

[1] Din acest moment trebuie să distingem între variabilele algoritmului **Alg** şi cele ale formulei $F_{Alg,D}$. Totodată, trebuie să observăm că fiecare literal din $F_{Alg,D}$ are o interpretare fixată în raport cu care probăm satisfiabilitatea formulei.

Formula **Start** este **CNF** şi este satisfiabilă dacă şi numai dacă execuția algoritmului începe la **t = 1** cu instrucțiunea a cărei etichetă este **1**.

Serial descrie execuția serială a oricărui flux de instrucțiuni din algoritmul **Alg**. La un moment de timp **t** este executată o singură instrucțiune a unui flux.

$$\mathbf{Serial} = \bigwedge_{t=1,p(n)} \Big(\Big(\bigvee_{i=1,l} S(i,t) \Big) \wedge \bigwedge_{\substack{i,j=1,l \\ i \neq j}} (\neg S(i,t) \vee \neg S(j,t)) \Big)$$

Formula arată că la orice moment **t** în cursul execuției unui flux de instrucțiuni al algoritmului există o instrucțiune executată şi aceasta este unică. Dacă la momentul **t** există două instrucțiuni **i≠j** executate într-un flux atunci termenul $\neg S(i,t) \vee \neg S(j,t)$ este fals, iar formula **Serial** nu este satisfăcută. Formula este în formă **CNF** şi este satisfiabilă doar în cazul execuției seriale a oricărui flux de instrucțiuni al algoritmului.[2]

Flow completează proprietatea de serialitate a execuției fluxurilor de instrucțiuni ale algoritmului. Pentru fiecare instrucțiune **i** a algoritmului **Alg** şi pentru fiecare moment de timp **t** din cursul execuției, **Flow** conține un termen care arată că:

- fie instrucțiunea **i** nu este executată la momentul **t**,
- fie **i** este executată la momentul **t**, iar instrucțiunea executată la momentul **t+1** este univoc determinată conform instrucțiunii **i**.

$$\mathbf{Flow} = \bigwedge_{\substack{t=1,p(n)-1 \\ i=1,l}} (\neg S(i,t) \vee Next_{i,t})$$

$Next_{i,t} = S(i,t+1)$, dacă instrucțiunea $S(i,t)$ este **success** sau **fail**. Prin urmare, după execuția unei astfel de instrucțiuni terminale se pretinde că algoritmul buclează pe instrucțiunea respectivă.

$Next_{i,t} = S(j,t+1)$, dacă instrucțiunea $S(i,t)$ este **goto j**;

$Next_{i,t} = (B(x,1,t) \wedge S(j,t+1)) \vee (\neg B(x,1,t) \wedge S(i+1,t+1))$, dacă instrucțiunea $S(i,t)$ este **if(x) goto j**. Atunci, dacă valoarea variabilei **x** din **Alg** este **adevărat** la momentul **t**, următoarea instrucțiune executată este **j**. Altfel algoritmul continuă cu instrucțiunea **i+1**. Deşi termenul $(\neg S(i,t) \vee Next_{i,t})$, cu $Next_{i,t}$ în forma de mai sus, nu este **CNF**, el poate fi convertit în formă **CNF** conform: $a \vee (b \wedge c) \vee (d \wedge e) = (a \vee b \vee d) \wedge (a \vee c \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee e)$.

$Next_{i,t} = S(i+1,t+1)$, dacă $S(i,t)$ nu este o instrucțiune de control **success**, **fail**, **goto**, **if...goto**.

Conform particularizărilor termenului $Next_{i,t}$ rezultă că **Flow** este **CNF** şi este satisfiabilă dacă şi numai dacă algoritmul are o execuție corectă în ceea ce priveşte determinarea următoarei instrucțiuni de executat.

---

[2] În prezența formulei **SERIAL**, formula **START** poate fi simplificată **START= S(1,1)**. Inițial, s-a preferat forma redundantă pentru o simplifica explicația.

Eval corespunde execuției efective a instrucțiunii de la momentul **t** şi modificării valorilor variabilelor **Var** ale algoritmului. Pentru fiecare instrucțiune **i** a algoritmului **Alg** şi pentru fiecare moment de timp **t** din cursul execuției, **Eval** conține un termen care arată că:

- fie instrucțiunea **i** nu este executată la momentul **t**,
- fie **i** este executată la momentul **t**, iar valoarea variabilelor **Var** la momentul **t+1** este cea corespunzătoare instrucțiunii **i** şi valorilor **Var** la momentul **t**.

$$\mathbf{Eval} = \bigwedge_{\substack{t=1,p(n)-1\\ i=1,l}} (\neg \mathbf{S(i,t)} \vee \mathbf{Eval_{i,t}})$$

Dacă instrucțiunea **S(i,t)** nu este o atribuire, atunci

$$\mathbf{Eval_{i,t}} = \bigwedge_{\substack{x \in Var\\ j=1,w}} ((\mathbf{B(x,j,t)} \wedge \mathbf{B(x,j,t+1)}) \vee (\neg \mathbf{B(x,j,t)} \wedge \neg \mathbf{B(x,j,t+1)}))$$

În acest caz, formula $\mathbf{Eval_{i,t}}$ este satisfiabilă dacă şi numai dacă variabilele algoritmului îşi păstrează nealterate valorile atunci când nu sunt modificate explicit. Termenul $\neg \mathbf{S(i,t)} \vee \mathbf{Eval_{i,t}}$ nu este în formă **CNF**, dar poate fi transformat conform echivalenței $\mathbf{a \vee (b \wedge c) \vee (\neg b \wedge \neg c) = (a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c)}$.

Dacă **S(i,t)** este o atribuire de forma $\mathbf{x = choice(\{V_1, V_2, ..., V_k\})}$, unde $\mathbf{V_i}$, $\mathbf{i=1,k}$, sunt variabile (scalare) din **Alg**, atunci la momentul **t+1**:

- variabilele din **Var** diferite de **x** îşi păstrează valorile de la momentul **t**;
- variabila **x** poate avea orice valoare desemnată de variabilele $\mathbf{V_j}$, $\mathbf{j=1,k}$.

În acest caz, formula $\mathbf{Eval_{i,t}}$ nu reflectă în mod explicit construcția copiilor algoritmului. Esențială este descrierea posibilității de a lega variabilele algoritmului la toate valorile corecte, inclusiv cele specificate în instrucțiunile **choice**. Este ca şi cum întregul arbore al execuției algoritmului **Alg**, arbore ale cărui noduri corespund instrucțiunilor **choice**, ar fi aplatizat, fiecare instrucțiune din **Alg** fiind eligibilă pentru execuție în egală măsură. Perspectiva oarecum stranie este posibilă, deoarece formula construită nu descrie execuția unei anumite copii a algoritmului, ci doar restricțiile ce trebuie îndeplinite pentru execuția corectă a oricărei copii, deci practic a tuturor copiilor.

$$\mathbf{Eval_{i,t}} = \bigwedge_{\substack{y \in Var\\ y \neq x\\ j=1,w}} ((\mathbf{B(y,j,t)} \wedge \mathbf{B(y,j,t+1)}) \vee (\neg \mathbf{B(y,j,t)} \wedge \neg \mathbf{B(y,j,t+1)})) \wedge \bigvee_{r=1,k} \bigwedge_{j=1,w} ((\mathbf{B(V_r,j,t)} \wedge \mathbf{B(x,j,t+1)}) \vee (\neg \mathbf{B(V_r,j,t)} \wedge \neg \mathbf{B(x,j,t+1)}))$$

Formula $\neg \mathbf{S(i,t)} \vee \mathbf{Eval_{i,t}}$ nu este **CNF**, dar poate fi adusă la forma **CNF**.

Dacă **S(i,t)** este o atribuire **x = expresie**, unde **expresie** nu este **choice**, atunci $\mathbf{Eval_{i,t}}$ este o formulă care explicitează restricțiile ce trebuie îndeplinite de valorile variabilelor din **expresie** şi de **B(x,j,t+1)**, **j=1,w**, astfel încât operațiile din expresie să fie corecte. Funcționalitatea expresiei este descrisă prin precondiții şi postcondiții. Ca exemplu simplu, considerăm că instrucțiunea **S(i,t)** este atribuirea **x = y ∧ z** în urma căreia **x** ia valoarea **adevărat** la momentul **t+1** dacă şi numai dacă valorile variabilelor **y** şi **z** sunt **adevărat** la momentul **t**, adică $(y^t \wedge z^t \Rightarrow x^{t+1}) \wedge (x^{t+1} \Rightarrow y^t \wedge z^t)$. Doar bitul **1** al valorii lui **x** trebuie să fie determinat la momentul **t+1**; ceilalți biți ai lui **x** pot avea orice valori. În acest caz forma **CNF** a lui $\mathbf{Eval_{i,t}}$ este:

$$\mathbf{Eval_{i,t}} = (\neg B(y,1,t) \vee \neg B(z,1,t) \vee B(x,1,t+1)) \wedge (B(y,1,t) \vee \neg B(x,1,t+1)) \wedge (B(z,1,t) \vee \neg B(x,1,t+1)) \wedge \bigwedge_{\substack{y \in Var \\ y \neq x \\ j=1,w}} ((B(y,j,t) \vee \neg B(y,j,t+1)) \wedge (\neg B(y,j,t) \vee B(y,j,t+1)))$$

Pentru operații aritmetice dificultatea sintezei lui $\mathbf{Eval_{i,t}}$ creşte substanțial. Ocolim aici construcțiile care arată există formule $\mathbf{Eval_{i,t}}$ pentru operațiile aritmetice uzuale şi pentru compunerea lor. Toate aceste formule pot fi aduse la forma **CNF** şi îşi păstrează lungimea polinomială în raport cu **n** (mărimea datelor **D**).

Construcția unui termen **Eval** arată că formula rezultată este **CNF** şi este satisfiabilă dacă şi numai dacă modificarea valorilor variabilelor lui **Alg** este corectă în raport cu instrucțiunea executată în fluxul generic de instrucțiuni al algoritmului.

Stop corespunde terminării cu succes a algoritmului, deci execuției uneia din instrucțiunile **success** din **Alg**. Să presupunem că etichetele instrucțiunilor **success** sunt: $\mathbf{e_1, e_2, ..., e_r}$.

$$\mathbf{Stop} = \bigvee_{i=1,r} S(e_i, p(n))$$

Formula **Stop** este satisfiabilă dacă şi numai dacă la momentul **p(n)** se execută o instrucțiune **success**. De remarcat că instrucțiunea **success** poate fi executată prima dată la un moment de timp **t < p(n)**. Conform construcției formulei **Flow**, algoritmul continuă să execute instrucțiunea până la momentul **p(n)**.

Conform construcției celor şase tipuri de formule, rezultă că $\mathbf{F_{Alg,D}}$ este **CNF** şi este satisfiabilă dacă şi numai dacă **Alg(D)** are un flux corect de instrucțiuni care conduce la terminarea cu succes a algoritmului. De asemenea, formulele pot fi construite în timp polinomial, funcție de **p(n)**, pornind de la **Alg**. Rezultă că problema $\mathbf{Q_{Alg} \in NP}$ rezolvată de algoritmul **Alg** satisface $\mathbf{Q_{Alg} \leq_p SAT}$. ■

Un exemplu de reducere $Alg(D) \leq_p SAT$

Ca exemplu de construcție a formulei $F_{Alg,D}$ să considerăm algoritmul de mai jos, unde variabilele **x** şi **y** sunt booleene şi au un singur bit. Algoritmul decide dacă măcar una din variabilele **x** şi **y** este **1** (realizează calculul $x \vee y$). Datele algoritmului sunt **D=(1,0)**, **n=2**, timpul de execuție al algoritmului este **p(n)=3**, iar numărul de instrucțiuni este **l=4**.

```
Alg(x,y) {
  1: x = choice{x,y};
  2: if(x) goto 4;
  3: fail;
  4: success;
}
```

Pentru datele **D**, **x=1** şi **y=0**, formulele din $F_{Alg,D}$ sunt construite ca în demonstrația teoremei lui Cook şi, pentru claritate, nu sunt toate în formă **CNF**.

$$INIT = B(x,1,1) \wedge \neg B(y,1,1)$$
$$START = S(1,1)$$
$$STOP = S(4,3)$$

$$SERIAL = \bigwedge_{t=1,3} (S(1,t) \vee S(2,t) \vee S(3,t) \vee S(4,t)) \wedge (\neg S(1,t) \vee \neg S(2,t)) \wedge (\neg S(1,t) \vee \neg S(3,t)) \wedge (\neg S(2,t) \vee \neg S(3,t))$$

$$FLOW = \bigwedge_{t=1,2} (\neg S(1,t) \vee S(2,t+1)) \wedge (\neg S(2,t) \vee B(x,1,t) \wedge S(4,t+1) \vee \neg B(x,1,t) \wedge S(3,t+1)) \wedge (\neg S(3,t) \vee S(3,t+1)) \wedge (\neg S(4,t) \vee S(4,t+1))$$

$$EVAL = \bigwedge_{t=1,2} [\neg S(1,t) \vee (B(x,1,t) \wedge B(x,1,t+1) \vee \neg B(x,1,t) \wedge \neg B(x,1,t+1) \vee B(y,1,t) \wedge B(x,1,t+1) \vee \neg B(y,1,t) \wedge \neg B(x,1,t+1)) \wedge (B(y,1,t) \wedge B(y,1,t+1) \vee \neg B(y,1,t) \wedge \neg B(y,1,t+1)] \wedge [\neg S(2,t) \vee (B(x,1,t) \wedge B(x,1,t+1) \vee \neg B(x,1,t) \wedge \neg B(x,1,t+1)) \wedge (B(y,1,t) \wedge B(y,1,t+1) \vee \neg B(y,1,t) \wedge \neg B(y,1,t+1))] \wedge [\neg S(3,t) \vee (B(x,1,t) \wedge B(x,1,t+1) \vee \neg B(x,1,t) \wedge \neg B(x,1,t+1)) \wedge (B(y,1,t) \wedge B(y,1,t+1) \vee \neg B(y,1,t) \wedge \neg B(y,1,t+1))]$$

Pentru a studia satisfiabilitatea formulei $F_{Alg,D} = Init \wedge Start \wedge Serial \wedge Flow \wedge Eval \wedge Stop$ folosim notația:

$$\text{—}\text{—} formula\ (t{=}k) \rightarrow legări\ variabile\ din\ F_{Alg,D}$$

Notația indică ce legări ale diverselor variabile de forma **B(x,i,t)**, **B(y,i,t)** şi **S(i,t)** sunt necesare pentru a satisface sub-***formula*** din $\mathbf{F_{Alg,D}}$ pentru momentul de timp **t=k** al execuției algoritmului **Alg** cu datele **D**. Legările indicate țin seama de toate legările deja deduse ale variabilelor din $\mathbf{F_{Alg,D}}$.

—INIT→ B(x,1,1)=1, B(y,1,1)=0
—START→ S(1,1)=1
—SERIAL (t=1)→ S(2,1)=S(3,1)=S(4,1)=0
—EVAL (t=1)→ B(x,1,2)=0 *sau* **B(x,1,2)=1, B(y,1,2)=0**
—FLOW (t=1)→ S(2,2)=1
—SERIAL (t=2)→ S(1,2)=S(3,2)=S(4,2)=0

Varianta 1. **B(x,1,2)=0**

—FLOW (t=2)→ S(3,3)=1
—EVAL (t=2)→ B(x,1,3)=0, B(y,1,3)=0
—SERIAL (t=3)→ S(1,3)=S(2,3)=S(4,3)=0
—STOP→ 0

$\mathbf{F_{Alg,D}=0}$ pentru fluxul de instrucțiuni **1,2,3** executat pentru copia algoritmului în care **x=0** după execuția instrucțiunii **choice**. Legarea variabilelor este:

B(x,1,1) = 1
B(x,1,2) = B(x,1,3) = B(y,1,1) = B(y,1,2) = B(y,1,3) = 0
S(1,1) = S(2,2) = S(3,3) = 1
S(4,1) = S(4,2) = S(4,3) = 0
S(1,2) = S(1,3) = S(2,1) = S(2,3) = S(3,1) = S(3,2) = 0

Varianta 2. **B(x,1,2)=1**

—EVAL (t=2)→ B(x,1,3)=1, B(y,1,3)=0
—FLOW (t=2)→ S(4,3)=1
—SERIAL (t=3)→ S(1,3)=S(2,3)=S(3,3)=0
—STOP→ 1

$\mathbf{F_{Alg,D}=1}$ pentru fluxul de instrucțiuni **1,2,4** executat pentru copia algoritmului în care **x=1** după execuția instrucțiunii **choice**. Legarea variabilelor este:

B(x,1,1) = B(x,1,2) = B(x,1,3) = 1
B(y,1,1) = B(y,1,2) = B(y,1,3) = 0
S(1,1) = S(2,2) = S(4,3) = 1
S(3,1) = S(3,2) = S(3,3) = 0
S(1,2) = S(1,3) = S(2,1) = S(2,3) = S(4,1) = S(4,2) = 0

Se verifică propoziția: $\mathbf{F_{Alg,D}}$ este satisfiabilă dacă şi numai dacă există o cale terminată cu succes în arborele execuției algoritmului **Alg** pentru datele **D**.