Clasa S

1. Definitii
2. Teorema lui Savitch
3. Clasele PSPACE si NPSPACE
4. PSPACE-completitudine

Clasa S

<u>Definitie 1</u>

Fie M o MT determinista cu 1! banda, decidenta,

$w \in \Sigma^*$ un cuvant de intrare,

$C = C_0 C_1 C_2 \ldots C_d C_f$ un calcul oarecare efectuat de M pe intrarea w;

Complexitatea spatiu a M pe intrarea w este numarul intreg notat

$SPACE_M(w) = \min \{SPACE_M(C) \mid C$ este un calcul efectuat de M pe intrarea w$\}$

= numarul minim de locaţii de pe banda de intrare scanate de M pe intrarea w.

Clasa S

Definitie 2

Fie M o MT determinista cu 1! banda decidenta.

Complexitatea spatiu (determinist) a masinii Turing M este o functie

$f : N \to N$, definita prin:

f(n) = numărul maxim de locaţii de pe banda de intrare scanate de M pentru orice secvenţa de intrare de lungime n, $\forall n \in N$.

Mai spunem că M rulează într-un spaţiu de memorie egal cu f(n).

Notatia 1

$$f(n) = \max \{SPACE_M(w) \mid w \in L(M) \subseteq \Sigma^n\} = SPACE_M(n)$$

Clasa S

<u>Definitie 3</u>

Fie N o MT nedeterminista cu 1! banda, decidenta,

$w \in L(M) \subseteq \Sigma^*$;

Complexitatea spatiu a N pe intrarea w este numarul intreg notat $NSPACE_N(w)$ care reprezinta cel mai mic numar de locatii de pe banda de intrare scanat de N, dintre toate ramurile de calcul care accepta w.

<u>Definitie 4</u>

Fie N o MT nedeterminista cu 1! banda, decidenta,

Complexitatea spatiu (nedeterminist) a N este o functie

$f : N \to N$, definita prin:

$f(n)$ = numărul maxim de locatii de pe banda de intrare necesar lui N pentru prelucrarea unui cuvant $w \in L(N) \cap \Sigma^n$, $\forall n \in N$.

<u>Notatia 2</u>

$f(n) = \max \{NSPACE_N(w) \mid w \in L(N) \cap \Sigma^n\} = NSPACE_N(n)$

Clasa S

Definiţia 5

Fie f : $N \rightarrow$ R+.

Definim clasele de complexitate *SPACE(f(n))* şi *NSPACE(f(n))* astfel:

SPACE(f(n)) = { L $\subseteq \Sigma^*$ | ($\exists$) o MT deterministă care decide asupra limbajului L într-un spaţiu *O*(f(n)) }

NSPACE(f(n)) = { L $\subseteq \Sigma^*$ | ($\exists$) o MT nedeterministă care decide asupra limbajului L într-un spaţiu *O*(f(n)) }

Clasa S

Exemplul 1

Urmatorul algoritm rezolva *SAT* cu spaţiu linear.

M_1 = “Fie secvenţa de intrare <ϕ>, $\forall \phi$ = formulă booleeană:

1. Pentru fiecare combinaţie de valori de adevăr atribuite variabilelor booleene $x_1, x_2, \ldots, x_m$ din ϕ se executa Pasul 2:
   2. Se evaluează valoarea de adevăr a formulei ϕ.
3. Dacă ϕ se evaluează la 1, atunci M_1 acceptă; altfel, respinge.”

Clasa S

Evaluam complexitatea spatiu a M_1:

la fiecare iteraţie a etapei a 2-a, M_1 trebuie să memoreze numai combinaţia curentă de valori de adevăr ale variabilelor $x_1, x_2, \ldots, x_m$

=> M_1 utilizează aceeaşi porţiune a benzii de intrare

$\Rightarrow$ spaţiul necesar este de ordinul O(m).

Cum numărul de variabile m este mărginit superior de lungimea secvenţei de intrare $\rightarrow$

M_1 rulează în spaţiu de ordinul O(n).

Clasa S

<u>Exemplul 2</u>

Fie A∈AFN. Vrem să verificăm dacă el acceptă toate secvenţele din Σ*. Codificăm această problemă prin următorul limbaj

$ALL_{AFN} = \{ \langle A \rangle \mid A \in AFN \text{ şi } L(A) = \Sigma^* \}$

Clasa S

soluţie

N = “Fie secvenţa de intrare <A>, unde A este un AFN:

1. Se marchează starea iniţială a lui A.
2. Se repetă Pasul 3 de 2^q ori, unde $q = |Q|$.
   3. Se selectează nedeterminist un simbol de intrare şi se modifica pozitia marcajelor de pe starile lui A pentru a simula citirea acelui simbol.
4. Dacă la pasii 2 şi 3 apare o secvenţă pe care A o respinge (adică dacă la un moment dat nici unul dintre marcaje nu se afla pe o stare de acceptare a lui A) atunci N acceptă secvenţa de intrare <A>; altfel o respinge.”

Clasa S

demonstram ca N decide $\overline{ALL_{AFN}}$

Dacă există secvenţe din Σ^* pe care A le respinge

→ printre ele trebuie să se afle una de lungime cel mult 2^q , deoarece în cazul tuturor secvenţelor de lungime mai mare, respinse de A, poziţiile markerilor descrişi în algoritmul de mai sus ar trebui să se repete (A are prin ipoteză doar q stări).

Porţiunea din secvenţă cuprinsă între repetiţii poate fi eliminată pentru a se obţine astfel o secvenţă respinsă, mai scurtă.

=> N decide asupra limbajului.

calculam complexitatea spatiu a lui N

Algoritmul necesită spaţiu de memorie suplimentar numai pentru stocarea locaţiilor markerilor şi a contorului de ciclare

=> algoritmul necesită spaţiu linear

=> algoritmul rulează în spaţiu nedeterminist de ordinul *O(n).*

Clasa S

1. Definitii
2. Teorema lui Savitch
3. Clasele PSPACE si NPSPACE
4. PSPACE-completitudine

Clasa S

Unul dintre primele rezultate privind complexitatea spaţiu:

MT deterministe (**MTD**) pot simula **MT** nedeterministe (**MTN**) utilizând o cantitate de memorie surprinzător de mică.

O astfel de simulare pare să implice:

- din punct de vedere al complexităţii timp:

  o crestere exponentiala: de la f(n) la $2^{f(n)}$,

- din punct de vedere al complexităţii spaţiu:

  o creşterea polinomiala: de la f(n) la $f^2(n)$.

<u>Teorema lui SAVITCH</u>

Fie o funcţie f: $N \to R^+$ cu proprietatea: $f(n) \geq n$

$\Rightarrow$ *NSPACE (f(n))* $\subseteq$ *SPACE (f^2(n))*

Clasa S

demonstratie

vom rezolva o problemă mai generală, PROBLEMA TRANZITIEI:

fie două configuraţii $c_1 = uaq_rbv$ şi $c_2 = zcq_sdw$ $(a,b,c,d \in \Gamma, u,v,z,w \in \Gamma^*, q_r,q_s \in \mathbf{Q})$ ale unei **MTN** şi un număr $t \in \mathbf{N}$;

trebuie să verificăm dacă **MTN** poate trece din configuraţia c_1 în configuraţia c_2 în t paşi.

$\Rightarrow$ Rezolvând PROBLEMA TRANZITIEI pentru

c_1 = configuraţia de start a **MTN**;

c_2 = configuraţia de acceptare a **MTN**;

t = numărul maxim de paşi pe care îi poate face **MTN**.

putem verifica dacă **MTN** acceptă secvenţa de intrare primită.

Clasa S

- se caută o configuraţie intermediară c_m,
- se verifică recursiv dacă
  - c_1 trece în c_m în t/2 paşi,
  - c_m trece în c_2 în t/2 paşi.

Acest algoritm are nevoie de spaţiu pentru a memora informaţia din stiva recursivă.

Clasa S

Fiecare nivel al recurenţei utilizează pentru memorarea unei configuraţii un număr de locaţii de memorie de ordinul ***O***(f(n)).

Adâncimea recurenţei este log(t), unde t = timpul maxim utilizat de **MTN** pe o ramură oarecare de calcul.

Stim că t = $2^{\boldsymbol{O}(f(n))}$ $\Rightarrow$ log(t) = ***O***(f(n))

=> simularea deterministă necesită O(f(n)) . O(f(n)) = ***O***(f^2(n)) locaţii de memorie.`

inaltimea stivei spatiul pt memorarea unei config.

Clasa S

<u>(i) Definim procedura TRANZ</u>

Fie: w secvenţa de intrare primită de N,

t $\in$ N (pentru simplificare, putem presupune că t este o putere a lui 2),

c_1, c_2 două configuraţii oarecare ale N;

atunci, TRANZ(c_1, c_2, t) acceptă dacă N poate trece din configuraţia c_1 în configuraţia c_2 printr-un calcul nedeterminist cu <u>maximum</u> t paşi; altfel respinge.

Clasa S

TRANZ = “Fie datele de intrare c_1, c_2 şi t, ca mai sus:

1. Dacă $t = 1$ atunci se verifică direct dacă $c_1 = c_2$ sau dacă c_1 trece în c_2 într-un singur pas conform funcţiei de tranziţie a lui N.

Dacă oricare dintre cele două condiţii este îndeplinită atunci TRANZ acceptă; dacă nici una dintre condiţii nu e îndeplinită atunci TRANZ respinge.

2. Dacă $t > 1$ atunci următoarele instrucţiuni se execută pentru fiecare configuraţie c_m a lui N pe intrarea w şi cu spaţiu de lucru f(n):
   3. Se rulează TRANZ(c_1, c_m, t/2).
   4. Se rulează TRANZ(c_m, c_2, t/2).
   5. Dacă paşii 3 şi 4 se încheie ambii cu acceptare, atunci TRANZ acceptă.
6. Dacă cel puţin o dată cele 2 condiţii nu se îndeplinesc, atunci TRANZ respinge.”

Clasa S

(ii) Definim o **MTD** M care simulează N astfel:

- modificăm N astfel încât atunci când N acceptă, N îşi goleşte banda de intrare şi îşi deplasează cursorul în celula cea mai din stânga, intrând astfel într-o configuraţie numită c_{accept};
- notăm cu c_{start} configuraţia de start a lui N pe intrarea w ;
- alegem o constantă d =numar natural astfel încât, pe intrarea w, unde |w|=n, N să treacă prin maxim $2^{df(n)}$ configuraţii şi să utilizeze f(n) celule de pe banda de intrare.

=> ne asigurăm că $2^{df(n)}$ este o limită superioară pentru timpul de lucru al oricărei ramuri de calcul a lui N pe intrarea w.

M = "Fie cuvântul de intrare w:

1. Returnează rezultatul produs de TRANZ(c_{start}, c_{accept}, $2^{df(n)}$)."

Clasa S

<u>(iii) corectitudinea lui M</u>

algoritmul TRANZ rezolvă PROBLEMA TRANZITIEI

→ M simulează corect N.

Mai trebuie să demonstrăm că M lucrează cu spaţiu ***O***$(f^2(n))$.

<u>(iv) complexitatea lui M</u> (aratam ca M lucrează în spaţiu $O(f^2(n))$)

Procedura TRANZ se apelează pe ea însăşi recursiv

→ la fiecare apel, ea memorează în stivă

- numărul de ordine al pasului de calcul curent,
- valorile lui c_1, c_2, t pt a le invoca la revenirea din apelul recursiv.

=> fiecare nivel de recurenţă utilizează un spaţiu de lucru suplimentar de ordinul ***O***$(f(n))$.

Clasa S

În plus, la fiecare apel recursiv valoarea constantei t=nr. paşi = nr. config. necesare pentru calcularea w, se injumătăţeşte (cf. procedurii).

Iniţial, $t = 2^{df(n)}$

$\Rightarrow$ adâncimea stivei (nr de apeluri recursive necesare) este $O(\log_2 t)=\boldsymbol{O}(\log_2 2^{df(n)})=O(d.f(n))=\boldsymbol{O}(f(n))$

$\Rightarrow$ spaţiul de memorie total este $\boldsymbol{O}(f(n)).\boldsymbol{O}(f(n))=\boldsymbol{O}(f^2(n))$.

Clasa S

<u>(v) rezolvarea unei dificultati tehnice:</u>

la fiecare apel recursiv al procedurii TRANZ, algoritmul M trebuie să cunoască valoarea lui f(n).

Soluţia: modificam M astfel încât M să încerce pe rând diverse valori pentru f(n): f(n) = 1, 2, 3, …

Pentru fiecare valoare f(n) = i, algoritmul M modificat utilizează TRANZ pentru a verifica dacă N poate ajunge in final in configuratia c_{accept}, testand daca N ajunge in vreuna dintre configuratiile de lungime i+1.

Clasa S

De asemenea, M utilizează procedura TRANZ pentru a verifica dacă N foloseşte cel puţin <u>i+1</u> locaţii, astfel:

fie f(n) = i; algoritmul M modificat utilizează procedura TRANZ pentru a verifica dacă N poate ajunge din configuraţia de start în una dintre configuraţiile de lungime <u>i+1</u> astfel:

dacă nici una dintre configuraţiile de lungime <u>i+1</u> nu este atinsă atunci M respinge;

dacă este atinsă una dintre configuraţiile de lungime <u>i+1</u> şi aceasta este o configuraţie de acceptare atunci M acceptă;

dacă este atinsă una dintre configuraţiile de lungime <u>i+1</u> şi aceasta <u>nu</u> este o configuraţie de acceptare atunci M continuă cu f(n) = <u>i+1</u>.

Clasa S

(vi) algoritmul M modificat:

M' = " Fie secventa d eintrare $w \in \Sigma^*$:

1. Fie i=1.
   2. Fie f(n)=i.
   3. Fie c_α "prima" configuratie de lungime i+1 a lui N:
   4. Se ruleaza TRANZ(c_{start}, c_α, $2^{d.i}$).

      Daca TRANZ respinge si daca mai exista cel putin o configuratie de lungime i+1 netestata, atunci se reia Pasul 4.

      Daca TRANZ respinge si daca nu mai exista nicio configuratie de lungime i+1 netestata, atunci M' respinge.
   5. Daca TRANZ accepta si daca $c_{i+1}=c_{accept}$ atunci M' accepta.

      Daca TRANZ accepta si daca $c_{i+1} \neq c_{accept}$ atunci $i \leftarrow i+1$ si se reia Pasul 2."

Clasa S

1. Definitii
2. Teorema lui Savitch
3. Clasele PSPACE si NPSPACE
4. PSPACE-completitudine

Clasa S

<u>Definiţia 1</u>

Notăm cu ***PSPACE*** clasa limbajelor decidabile în spaţiu polinomial de către **MT** deterministe cu o singură bandă de intrare:

$$PSPACE = \bigcup_{k \in N} SPACE(n^k)$$

Notăm cu ***NPSPACE*** clasa limbajelor decidabile în spaţiu polinomial de către **MT** nedeterministe cu o singură bandă de intrare:

$$NPSPACE = \bigcup_{k \in N} NSPACE(n^k)$$

Clasa S

Observaţia 1

PSPACE = NPSPACE

(cf. Teoremei lui Savitch)

Exemplul 3

Cf. Ex.1: SAT $\in$ **SPACE(n);**

Cf. Ex.2: $\overline{ALL_{AFN}} \in NSPACE(n)$

clasele de complexitate spaţiu determinist sunt închise la complementară => $ALL_{AFN} \in$ **coNSPACE(n)**.

cf. Teorema Savitch => $ALL_{AFN} \in$ **SPACE(n²)**

=> SAT, $ALL_{AFN} \in$ **PSPACE**.

Clasa S

Relaţia dintre clasele de complexitate timp şi spaţiu

(i) *P* $\subset$ *PSPACE*

Fie funcţia t: $N \rightarrow N$, $t(n) \geq n$, $(\forall)$ $n \in N$;

orice MT care rulează în timp t(n) poate utiliza cel mult t(n) celule de pe banda de intrare deoarece la fiecare pas de calcul ea nu poate examina decât cel mult o celulă.

(ii) *NP* $\subset$ *PSPACE*

Cu un raţionament analog rezultă că *NP* $\subseteq$ *NPSPACE*.

Cf. Observaţiei 1: PSPACE = NPSPACE => *NP* $\subseteq$ *PSPACE*.

Clasa S

(iii) Reciproc, putem găsi o majorare pentru complexitatea timp a unei **MT** în funcţie de complexitatea spaţiu.

Cf. def. ant.: o configuratie a unui automat: C_i=uaq$_i$bv

Cf. lema ant: un automat linear mărginit (**ALM**) cu r stări, care citeşte un cuvânt de lungime n de pe banda de intrare şi care dispune de un alfabet de intrare Σ cu s elemente poate avea cel mult ***r.n.s^n*** configuraţii distincte.

=> o MT$\in$**PSPACE** (adica o MT care utilizează f(n) locaţii de memorie, unde f : N → N, f(n) $\geq$ n), poate avea cel mult ***f(n).2^{O(f(n))}*** configuraţii distincte.

Am pp ca **MT** se opreşte indiferent de secvenţa de intrare primită

$\Rightarrow$ MT nu cicleaza

=> pe parcursul oricarui calcul, nicio configuratie nu se repeta

Clasa S

$\Rightarrow$ o **MT** care utilizează f(n) locaţii de memorie trebuie să execute f(n)**.**$2^{O(f(n))}$ paşi de calcul, deci:

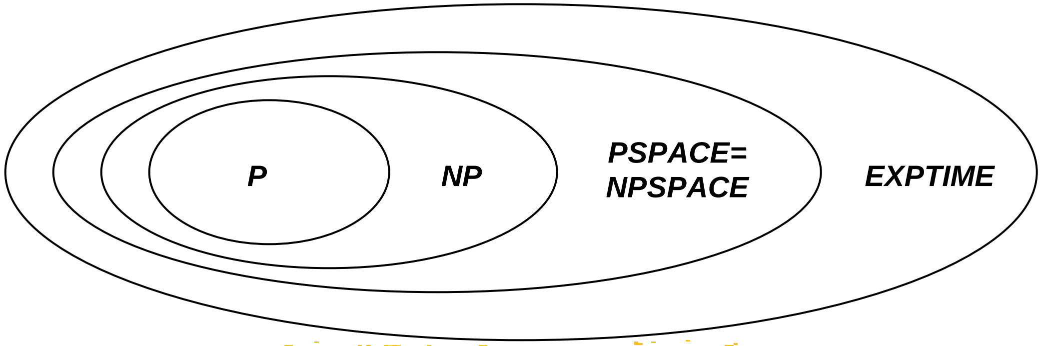
=> MT$\in$PSPACE ruleaza in timp cel mult f(n)**.**$2^{O(f(n))}$

=> MT $\in$EXPTIME =>

$$PSPACE \subseteq EXPTIME = \bigcup_{k \in N} TIME(2^{n^k})$$

Din (i), (ii) şi (iii) rezultă că:

P $\subseteq$ NP $\subseteq$ NPSPACE = PSPACE $\subseteq$ EXPTIME



Clasa S

<u>Observaţia 2</u>

Nu se ştie <u>încă</u> dacă vreuna dintre incluziuni nu este de fapt chiar o egalitate.

S-ar putea descoperi oricând o simulare asemănătoare celei din demonstraţia Teoremei lui Savitch care să permită fuzionarea unora dintre aceste clase într-o singură clasă.

Pe de altă parte, se demonstrează că ***P*** $\neq$ ***EXPTIME***

$\Rightarrow$ cel puţin una dintre incluziunile de mai sus este strictă dar nu se ştie <u>care</u>!!

În fapt, cei mai mulţi cercetători cred că <u>toate</u> incluziunile sunt stricte, adică acceptă diagrama de mai sus ca descriind cel mai corect relaţia dintre clasele de complexitate timp şi spaţiu:

Clasa S

1. Definitii
2. Teorema lui Savitch
3. Clasele PSPACE si NPSPACE
4. PSPACE-completitudine

Clasa S

Definitia 2

Un limbaj B este ***PSPACE*-complet** $\Leftrightarrow$

(1) B $\in$ ***PSPACE***

(2) ($\forall$) A $\in$ ***PSPACE*** $\rightarrow$ **A $\leq_P$ B** ($\forall$A este polinomial reductibil la B)

Dacă limbajul B satisface numai condiţia (2) atunci spunem că el este ***PSPACE*-dificil** (***PSPACE-hard***).

Clasa S

<u>Observaţia 3</u>

PSPACE-completitudinea utilizeaza tot notiunea de reductibilitate in **timp** polinomial. Motivul:

⇒ Regula este: oridecâteori definim probleme complete pentru o anumită clasă de complexitate, modelul in raport cu care definim reducerea trebuie să fie mai limitat (ca si complexitate) decât modelul folosit pentru definirea clasei de complexitate însăşi.

Clasa S

Exemple de probleme PSPACE-complete

Problema 1: TBQF

Definiţia 3

Formulă booleeană complet cuantificată =

o formulă booleeană în care fiecare variabilă este cuantificată.

Exemplul 4

$\phi = \forall x \ \exists y \ [(x \vee y) \wedge (\neg x \vee \neg y)]$

Definiţia 4

Problema TQBF constă în determinarea valorii de adevăr a unei formule booleene complet cuantificate oarecare şi este formalizată prin limbajul:

$TQBF = \{<\phi> \mid \phi$ este o formulă booleeană complet cuantificata $\}$

Clasa S

Teorema 2

TQBF este o problemă ***PSPACE***-completă.

ideea demonstraţiei

1. Pentru a demonstra că *TQBF* $\in$ ***PSPACE*** construim un algoritm care asignează valori de adevăr variabilelor din formulă şi apoi evaluează recursiv valoarea de adevăr a acesteia.

   Pe baza acestei informaţii, algoritmul poate determina dacă formula dată este sau nu adevărată.

2. Pentru a arăta că toate limbajele L din ***PSPACE*** se reduc polinomial la *TQBF:*

   (i) construim pentru L o MT, M, cu spaţiu de lucru polinomial,

   (ii) construim o reducere polinomială care asociază o secvenţa unei formule booleeane complet cuantificate ϕ care codifică o simulare a MT, M, pe acea intrare;

   (iii) formula este adevărată $\Leftrightarrow$ M acceptă.

Clasa S

<u>Problema 2: Strategii de castig pentru jocuri</u>

Fie o formulă booleeană complet cuantificată

$\phi = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_k [\psi]$;

considerăm un joc la care participă doi jucători, E şi U , care asignează <u>pe rând</u> valori de adevăr variabilelor $x_1, x_2, x_3, \ldots, x_k$ (in ordinea data de ordinea in care apar cuantificatorii) astfel:

- E asignează valori numai variabilelor cuantificate existenţial;
- U asignează valori numai variabilelor cuantificate universal.

Dacă prin această asignare formula ψ (partea lui ϕ care nu contine cuantificatori) se evaluează la TRUE, atunci câştigă jucătorul E; altfel câştigă jucătorul U .

Clasa S

Spunem că jucătorul E are o <u>strategie câştigătoare</u> pentru formula ϕ dacă – indiferent de asignările făcute de jucătorul U – jucătorul E poate asigna valori de adevăr varibilelor (legate prin cuantificatori existenţiali) în aşa fel încat să câştige.

Este evident că jucătorul E are o startegie câştigătoare pentru ϕ dacă şi numai dacă ϕ se evaluează la valoarea 1. Avem astfel:

<u>Teorema 3</u>

Limbajul *Formula-joc* = {<ϕ> | jucătorul E are o strategie câştigătoare pentru formula-joc asociată formulei booleene complet cuantificate ϕ} este ***PSPACE***-complet

Clasa S

Problema 3: Jocul geografic

Jocul geografic sau *Antakhshari*, se refera la numele oraşelor.

El poate fi modelat printr-o problemă de grafuri orientate:

- fiecare nod din digraf este etichetat cu numele unui oraş
- există un arc de la nodul u la nodul v dacă ultima literă a etichetei nodului u coincide cu prima literă a etichetei nodului v.

Jocul începe într-un nod oarecare, fixat; cei doi jucători se deplasează alternativ în digraf, de-a lungul arcelor, în noduri nevizitate încă. Jucătorul care nu reuşeşte să facă o nouă deplasare pierde.

Jocul poate fi generalizat la un digraf arbitrar (în care nodurile şi arcele nu mai au nici o legatura cu oraşele sau literele) şi un nod predeterminat.

Problema constă în a determina dacă primul jucător are o strategie câştigătoare. Avem astfel:

Clasa S

<u>Teorema 4</u>

Limbajul *Joc-geografic* = {<G,s> | primul jucător are o strategie câştigătoare pentru jocul *Antakhshari* generalizat în digraful G, dacă jocul începe din nodul s }

este *PSPACE*-complet.

ideea demonstraţiei

Pentru a demonstra teorema ar trebui să găsim o reducere în timp polinomial a problemei *Formula-joc* la problema *Joc-geografic*.

Intrucât se crede că $P \subset PSPACE$ este de presupus că nu există nici un algoritm cu timp de lucru polinomial care să-i permită primului jucător să <u>verifice</u> existenţa unei strategii câştigătoare, cu atât mai puţin să o <u>determine</u>.

Clasa S

1. Definitii
2. Teorema lui Savitch
3. Clasele PSPACE si NPSPACE
4. PSPACE-completitudine