# Inkrementális Tesztgenerálás FSM Modellhez

# Háttér: rendszermodell

- **Modell:**
  - **Tradicionális determinisztikus véges automata - FSM modell**
  - **Működést modellezi**
  - **Általános: M = I, O, S, f(s, i), ( $S_0$)**
    - » **Bemenet (input), kimenet (output) szimbólumok és állapotok véges halmazai,**
    - » **Állapotátmenet fv. -- f: S×I → $2^{O×S}$**
  - **Determinisztikus: M = I, O, S, λ(s, i), δ(s, i) ( $S_0$)**
    - » **Átmeneti (transition) függvény: δ: S × I → S**
    - » **Kimeneti (output) függvény : λ: S × I → O**

# Háttér: rendszermodell

- **Trace:**
  - **i/o szekvencia: $i_1o_1i_2o_2i_3o_3$**
- **Equivalence – állapot**
  - **Két állapot ekvivalens, ha minden input szekvenciára ugyan azt az output szekvenciát adja**
  - **Ha két állapot nem ekvivalens akkor létezik őket megkülönböztető „separating sequence”**
- **Equivalence – automata**
  - **Két automata ekvivalens, ha M minden állapotához megfeleltethető M’ legalább egy állapota**
  - **Az egymással ekvivalens automaták halmazában létezik egy minimál automata a lehető legkisebb állapotszámmal**

# Háttér: rendszermodell

- **Automata minimalizálás**
  1. **Lépcsős tábla**
  2. **Equivalencia halmazok használata:**
     - » **1. lépés: Output alapján.**
       - **$S_i$ és $S_j$ egy csoportban ha minden inputra ugyan az az outputjuk**
     - » **2. lépés: Next State alapján**
       - **$S_i$ és $S_j$ egy csoportban ha minden input esetén ugyan abba az ekvivalencia halmazba tartozik a next state-jük**
     - » **Iteráció amíg**
       - **Minden equivalencia halmaz egy elemű vagy nem lehet tovább bontani a halmazokat**
     - » **Példa később**

# Háttér: rendszermodell

- **Modell jellemzők:**
  - **Determinisztikus: Maximum egy átmenet lehetséges minden (s, i) párra**
  - **Teljesen specifikált: Legalább egy átmenet van minden (s, i) párra**
    - » **Együtt: Pontosan egy átmenet van minden (s, i) párra**
  - **Erősen összefüggő: Minden állapot elérhető bármely más állapotból**
  - **Minimál: Nincs két ekvivalens állapota**
    - » **Ekvivalens állapot: $s_1$ $s_2$ minden x input szekvenciára azonos output szekvenciát ad**
  - **Megbízható reset:**
    - » **Létezik egy $s_0$ kiinduló állapot és egy *r* bemenet szimbólum, amely az automata bármely állapotából $s_0$-ba visz. Megbízható, ha ez bármely hibás automatára is igaz.**

# Háttér: rendszermodell

- **Completeness assumptions**
- **Nem specifikált átmenetek kezelése – az automata teljesen specifikáltként kezelése**
  1. **Null (error) állapot és Null output**
     - **Nem specifikált transition null outputot ad és Null állapotba kerül**
  2. **Null (error) output**
     - **Nem specifikált transition null outputot ad és a kiinduló állapotban marad**
  3. **Don't Care**
     - **Nem specifikált transition random outputot random állapotba kerül**

# Háttér: tesztelési problémák

- **Konformancia tesztelés (megfelelőség vizsgálat, hiba detektálás)**
  - **Adott egy referencia automata – a specifikáció**
    - » **Fehér doboz (white box) – ismerjük a felépítését (átmenet, kimenet fv.)**
  - **Adott egy fekete doboz (black box) automata – az implementáció**
    - » **Csak a bemenet/kimenet viselkedését tudjuk megfigyelni**
  - **Vizsgáljuk meg, hogy az implementáció a specifikációnak megfelel-e.**
- **Azaz találjuk meg a hibákat**
- **Általában nem lehet megfelelőséget bizonyitani**
  - **„Testing can show the presence of bugs, but never their absence”**

# Háttér: tesztelési problémák

- A konformancia tesztelés probléma megoldható bizonyos feltételek mellett
- Bizonyíthatjuk a megfelelőséget
  - Feltételek három csoportja:
    - » Specifikációval kapcsolatos feltevések
    - » A megfelelőséggel kapcsolatos feltevések – konformancia relációk
    - » Implementációval kapcsolatos feltevések – hibamodellek

1. A specifikáció
   - determinisztikus
   - erősen összefüggő
   - minimál automata
2. Mi az hogy megfelelő? – Konformancia relációk
   - Ebben az esetben az automaták ekvivalenciája

# Háttér: tesztelési problémák

- **Az implementációval kapcsolatos feltevések:**
  - **nem változik a teszt alatt**
  - **megegyező bemeneti szimbólumhalmazzal rendelkezik**
  - **Nincsen több állapota mint a specifikációnak**
  - **Vagy Hibamodell !**
    - » **Lényege: teszt készítésekor figyelembe vett hibákat definiálja**
    - » **Tehát: a figyelembe vett lehetséges implementációk számát korlátozza**
    - » **Pl: Nem lehet nem-determinisztikus az implementáció**

# Hibamodell

- **Jellemzően két vagy három fajta hibát szoktak FSM-ek esetén figyelembe venni:**
  - **Kimenet hiba**
    - » **Egy adott állapot, bemenet kombinációra az implementáció más kimenetet ad mit azt a kimeneti függvény előírja.**
    - » $\lambda_{spec}(s, i) \neq \lambda_{impl}(s, i)$
  - **Transition faults**
    - » **Egy adott állapot, bemenet kombinációra az implementáció más állapotba jut, mint azt az átmeneti függvény előírja.**
    - » $\delta_{spec}(s, i) \neq \delta_{impl}(s, i)$
  - **+1 – Állapot hibák**
    - » **+ állapot**
    - » **- állapot**

# Conformance teszt összefoglaló

- **Alapvetően hasonló struktúra:**
  - **Jussunk el minden transition kezdőállapotába**
  - **Teszteljük le a kimeneti függvényt (transition inputjára adott választ)**
  - **Teszteljük le a next state függvényt**
- **A TT method az utolsó lépéssel nem foglalkozik**
  - **Next state hiba maradhat**
- **A különbségek lényege a next state ellenőrzés módszere**

# Conformance teszt összefoglaló

- Next state ellenőrzés

1. Separating family of sequences
   - Minden $s_i$ állapothoz egy $Z_i$ elválasztó szekvencia halmaz
   - Az $s_i$ hez tartozó $Z_i$ halmaz minden más $s_k$ állapothoz tartalmaz egy x elválasztó szekvenciát
   - x (legalább prefixként) szerepel $Z_i$-ben és $Z_k$-ban is.
   - Minden minimál, completely specified, determinisztikus FSM-hez lehet ilyen szekvencia halmazokat generálni
2. Characterizing sequences (W-set)
   - Speciális eset: Minden $Z_i$ egyforma
3. UIO sequences
   - Speciális eset: Minden $Z_i$ egy elemű

# Conformance teszt összefoglaló

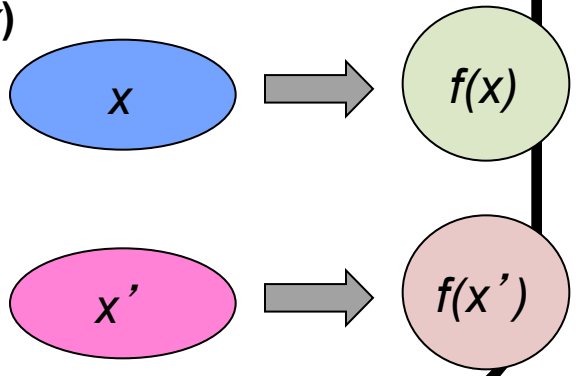
4. **Distinguishing Sequences**
   - **Speciális eset: Minden $Z_i$ egyforma és egy elemű**
5. **Status message**
   - **Speciális eset: Minden $Z_i$ egyforma és egy elemű; ez az egy elem egy hosszú és az állapot nem változik**

# Háttér: A HIS Method

- **Feltételezés: FSM megbízható resettel :**
  - **W, Wp, HIS – alapvető felépítés hasonló**
- **HIS method:**
  - **W-set helyett separating family of sequences**
  - **Állapot elérés: A *prefix-closed state cover set* – egy $q_i$ szekvencia minden $s_i$ állapothoz, mely elviszi az automatát $s_0$-ból $s_i$-be**
    - » **Prefix-closed: Legyen $K \subseteq A^*$ az A szimbólumhalmazon értelmezett szekvencia halmaz**
    - » **Legyen Pref(K) a K összes prefix szekvenciáját tartalmazó halmaz**
    - » **K prefix-closed ha Pref(K) = K, azaz K tartalmazza az összes prefix szekvenciát is.**
  - **Next state ellenőrzés: A separating family of sequences (family of harmonized identifiers) – Minden $s_i$ állapothoz rendel egy Zi halmazt, mely tartalmazza az $s_i$-a többi állapottól elválasztó szekvenciákat:**
- **Két fázis:**
  - **State identification: $r \cdot q_i \cdot Z_i$, minden állapotra**
  - **Transition testing: $r \cdot q_j \cdot i \cdot Z_k$, a maradék transitionra**
- **Komplexitás: $O(pn^3)$,**
  - **$pn^2$ darab 2n-nél rövidebb szekvencia, resettel elválasztva**

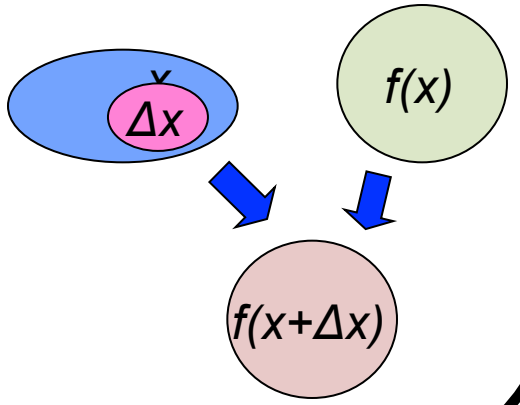
# Probléma: Változó specifikációk esetén rossz hatékonyság

- A létező tesztgeneráló algoritmusok statikus specifikációkat feltételeznek.
- Probléma: A tesztek mindig nulláról készülnek – még kis változás után is
  - Azaz „batch” algorithmusok
- Batch működés:
  - Adott: FSM rendszer specifikáció (input *x*), checking sequence (output *f(x)*)
  - A rendszer specifikáció változik (*x’* = *x*+*Δx*)
    - Új checking sequence (Új output – f(x’)) kizárólag az új FSM (x’) alapján
    - Korábbi checking sequence kidobva



# Cél: Hatékony Tesztgenerálás változó specifikációk esetére

- **Megfigyelés:**
  - **A specifikációk változnak, fejlődnek**
  - **Új követelményeknek felel meg és új képességekkel bír**
  - **Feltételezhetjük, hogy volt korábbi rendszer verzió és ahhoz létezett checking sequence**
- **Cél: Új tesztgeneráló algoritmus mely képes a változásokat hatékonyan kezelni.**
  - **Használja fel a korábbi checking sequence által meglévő plusz információt**
  - **Az algoritmus komplexitása a változás nagyságától függjön ne pedig a rendszer komplexitásától**
  - **→ Magyarul: Inkrementális tesztgeneráló algoritmus**



# FSM változás modell

- A változásokat edit operátorokkal modellezhetjük
  - $\omega^M: T \rightarrow T$
  - „Atomic” operatorok
  - Tradicionális FSM hibamodellek alapján
- Determinisztikus FSM alapján
  - Next state, output change operatorok
- Két azonos állapotszámú FSM-hez mindig van egy editoperátor szekvencia mely az egyiket átalakítja a másikká

# Inkrementális tesztgeneráló algoritmus – követelmények

- **Input:**
  - **Kiinduló FSM M**
  - **Checking sequence M-hez**
  - **Változás mely M-et M'-vé alakítja**
- **Output:**
  - **Checking sequence M'-höz**
- **Minőségi követelmény:**
  - **M' minden átmenetére tesztelje az output és a next state függvényeket**
- **Hatékonysági követelmény:**
  - **Csak akkor generáljunk új tesztesetet, ha „szükséges"**
  - **Azaz ha a változás után nem teljesíti a célját**

# Inkrementális tesztgeneráló algoritmus – egy megfigyelés

- **A HIS-method alapvetően két független algoritmus szuperpozíciója:**
  1. **Prefix-closed state cover set generálás: Eljussunk minden állapotba**
  2. **Separating fam. of seq. létrehozása: Az átmenetek next state fv. ellenőrzése.**
- **Következmény:**
  - **Az inkrementális tesztgeneráló algoritmus is két autonom algoritmusból áll**
  - **Egy adott változás különbözőképp hathat a két algoritmusra!**

# Inkrementális tesztgeneráló algoritmus – két sub-algoritmus

- **A prefix-closed state cover set karbantartása:**
  - » **Cél: Egy valid prefix-closed state cover set létrehozása M'-höz.**
  - – **A probléma visszavezethető az állapot-átmeneti gráf egy feszitőfájának karbantartására.**
- **A separating family of sequences karbantartása**
  - » **Cél: Új separating family of sequences létrehozása M'-höz.**
  - – **A probléma alapvetően az állapotpárok halmazán értelmezhető nem az állapotok halmazán!**
  - – **Az állapotpárok halmazán bevezetünk egy segédgráfot, mely az elválasztó szekvenciákat reprezentálja**
  - – **A probléma visszavezethető a segédgráf feszítő erdőjének karbantartására.**

# Inkrementális tesztgeneráló algoritmus – struktúra

- **A két inkrementális algoritmus felépítése hasonló:**
  - **Mind a kettő irányított címkézett gráfokon értelmezett algoritmus**
- **Fő lépések:**

**1a) A változás végrehajtása**

- » **A módosított elemek azonosítása (állapotok / állapotpárok)**
- » **A vátozások végrahajtása az állapotátmenet gráfon / az állapotpárokon értelmezett segédgráfon**

**1b) A változás hatásainak megállapítása**

- » **Az érintett elemek meghatározása: állapotok / állapotpárok amelyek esetén a kimenet változik, azaz a kapcsolódó teszt eseteket módosítani kell**

**2) Új output létrehozása**

- » **Új teszt szekvenciák csak az érintett elemekhez**
- » **State-cover/Elválasztó szekvenciák**

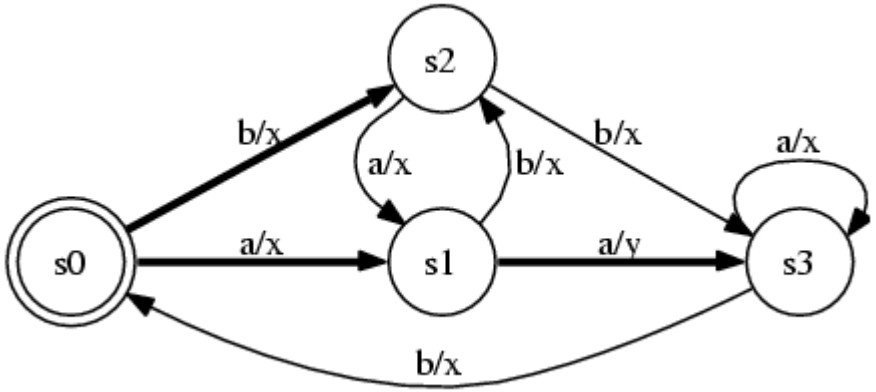
# A prefix-closed state cover set karbantartása 1/2

- 1a: Változtatás végrehajtás – triviális (1 változtatás)
- 1b:
  - A prefix-closed state cover set az M egy feszítőfája, mely a root-ból kifele irányított
  - Output change: nincs struktúrális változás → Nincs további érintett
  - Next state change
    - » Ha az értintett transition nem része az M spanning tree-jének akkor → Nincs  további érintett
    - » Ha az érintett  transition része a spanning treenek akkor az alatta levő subtree érintett

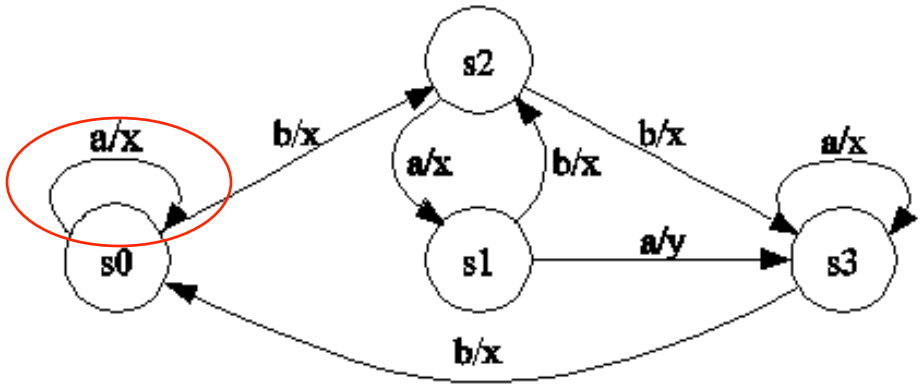
# A prefix-closed state cover set karbantartása 2/2

- 2: Új state cover szekvencia az érintettekez
  - A feszítőfa kiegészítése
    - » Ha a meglévő fából indulunk akkor nem teljesül a feltétel, miszerint csak a változástól függhet a komplexitás
  - Trükk: Visszafele keresünk: Az érintettekbe érkező transitionokon keresünk nem érintettekbe vezető élet
    - » Mivel az érintettekből $|érintett_Q| * p$, $p= |I|$ transition indul ezért $|érintett_Q| * (p+1)$ lépésben találunk utat nem érintettből érintettbe (ha létezik).
    - » Ezért a komplexitás $O(|érintett_Q| * p)$ ahol $1 \leq |érintett_Q| \leq n$

# Példa



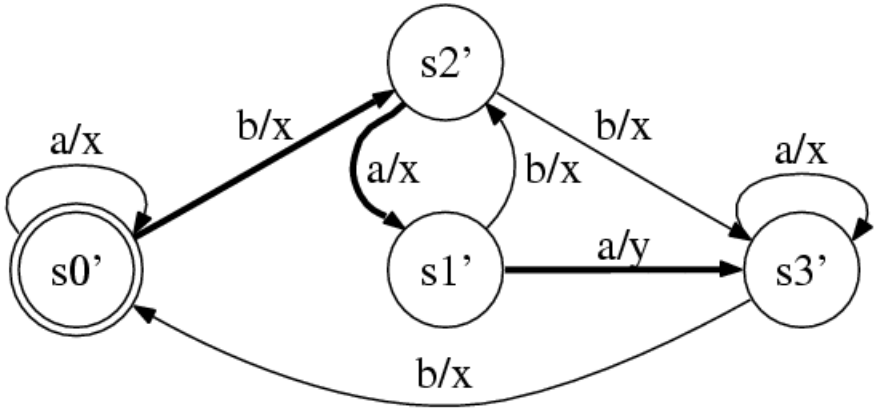

Eredeti FSM M




**Változás végrehajtása:**

$\omega(s_0, a, x, s_1) = (s'_0, a, x, s'_0)$

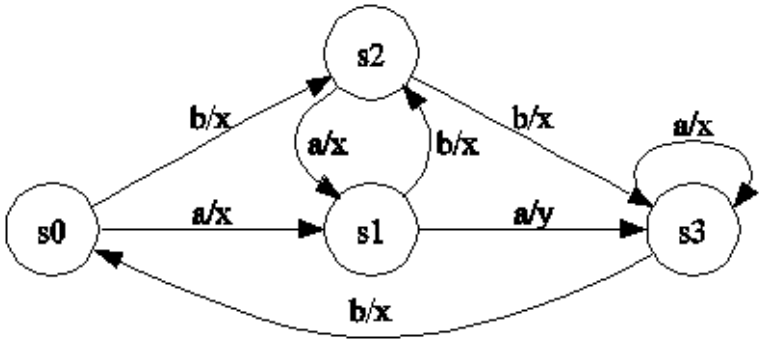
- **Érintett**
  - **s1, s3**
  - **(s0, a) és (s1, a) kikerül a feszítőfából**
  - **Pl. s1-ből visszafele keresünk, megtaláljuk a nem érintett s2-t**
  - **(s2,a) bekerül**
  - **Ugyanez s3-ra**
  - **(s1, a) bekerül**



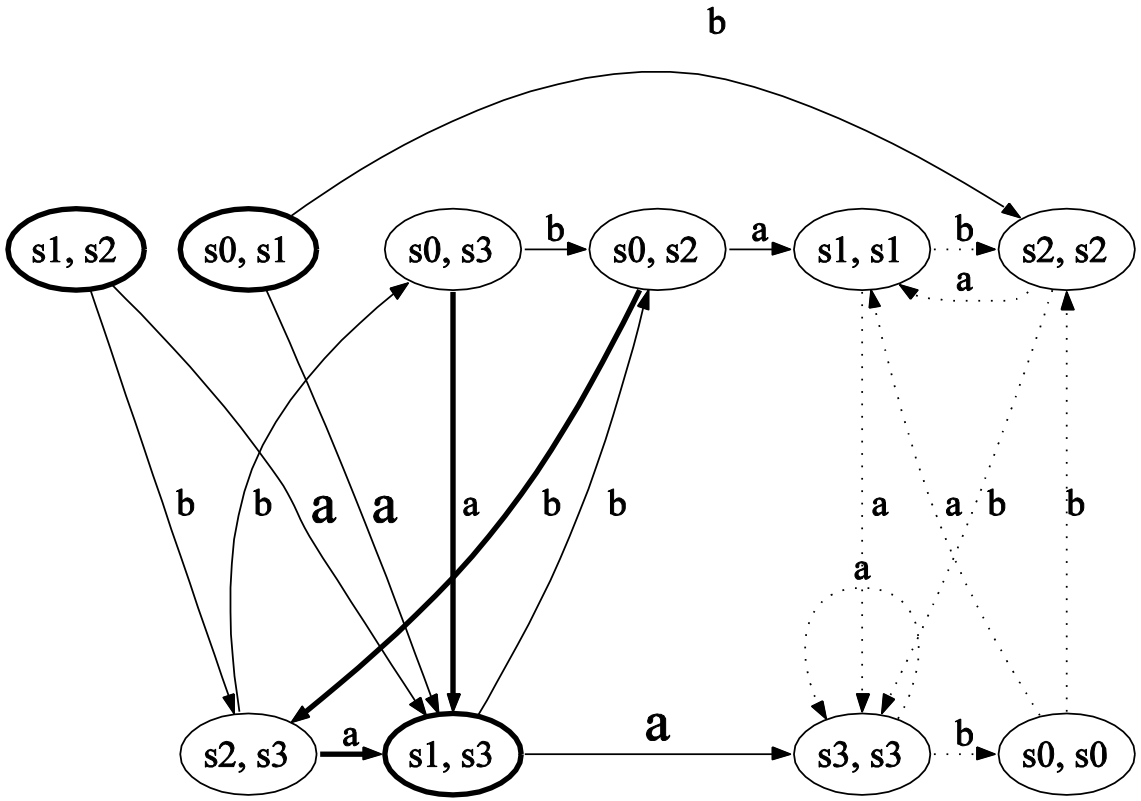
# A separating family of sequences karbantartása 1/4

- Hasonlóképp próbáljuk megoldani
- Segédgráf $A^M$ M-hez:
  - n(n+1)/2 csomópont, egy db az M minden $(s_j , s_k)$ állapotpárjához
  - i inputtal címkézett irányított él van $(s_j , s_k)$-ból $(s_l, s_m)$-be akkor és csak akkor ha $\delta(s_j , i) = s_l$ and $\delta(s_k, _i) = s_m$ az M automatában
- Az $A^M$-en ábrázolni tudjuk a separating sequenceket
  - Ha a separating sequencek prefix closed-ek akkor az $A^M$ feszítő erdőjét alkotják (root felé irányított)
  - Azonos állapotokat tartalmazó párok $(s_i, s_i)$ nem számítanak

# Példa




Eredeti FSM M




**Segédgráf:**

- **Nagyobb él címke:**
  - **“Separating input”:**
  - **Az adott állapotpárt megkülönböztető input**
- **Vastagított ellipszis:**
  - **“Separating állapotpárok”:**
  - **Állapotpárok separating inputtal**
- **Vastagított élek:**
  - **“Feszítő erdő”:**
  - **Separating állapotpárba vezető utak**
- **Separating sequence:**
  - **A vastagított élek inputjai + a separating input**
- **Dotted edges:**
  - **Azonos állapotpárok közötti élek**
  - **Irrelevánsak**

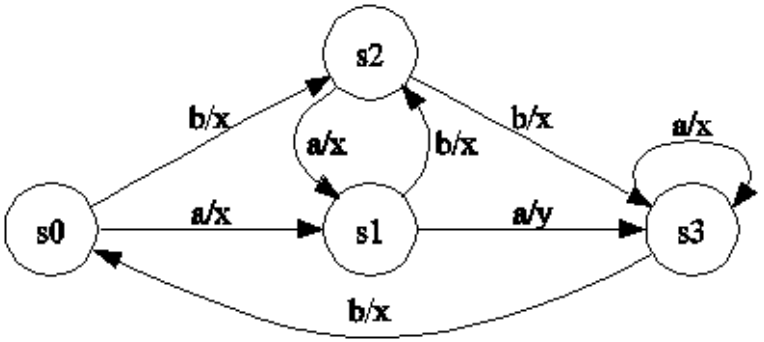
# A separating family of sequences karbantartása 3/4

- Innentől a megoldás hasonló a state cover esethez
- 1a) A változás végrehajtása
  - 1 módosítás -> n változás a AM-ben
- 1b) A változás hatásainak megállapítása
  - Output change:
    - » Ha separating state pár volt és az adott inputra már nem az akkor a hozzá tartozó tree érintett
    - » Különben nincs  további érintett
  - Next state change
    - » Ha az értintett transition nem része a spanning forestnek akkor nincs  további érintett
    - » Ha az érintett  transition része spanning forestnek akkor a hozzá tartozó subtree érintett

# A separating family of sequences karbantartása 4/4

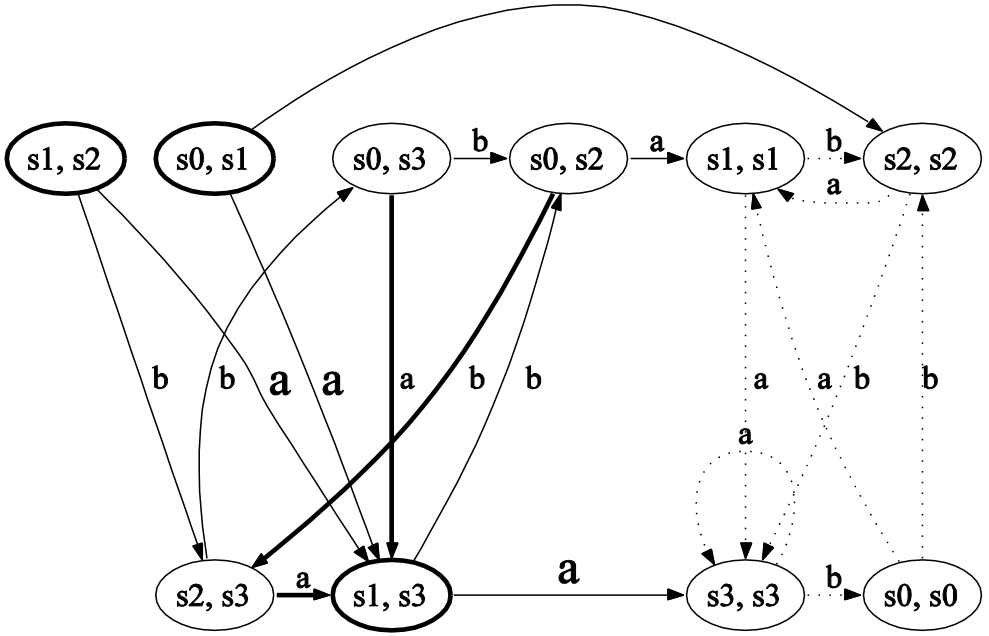
- 2: Új elválasztó szekvencia az érintettekez
  - A spanning forest kiegészítése
  - Itt az érintett állapotpárokból „előre” az élek irányáb keresünk, és kiegészítjük a spanning forestet
- Komplexitás $O(|érintett_Z| * p)$ ahol $n \leq |érintett_Z| \leq n^2$

# Példa

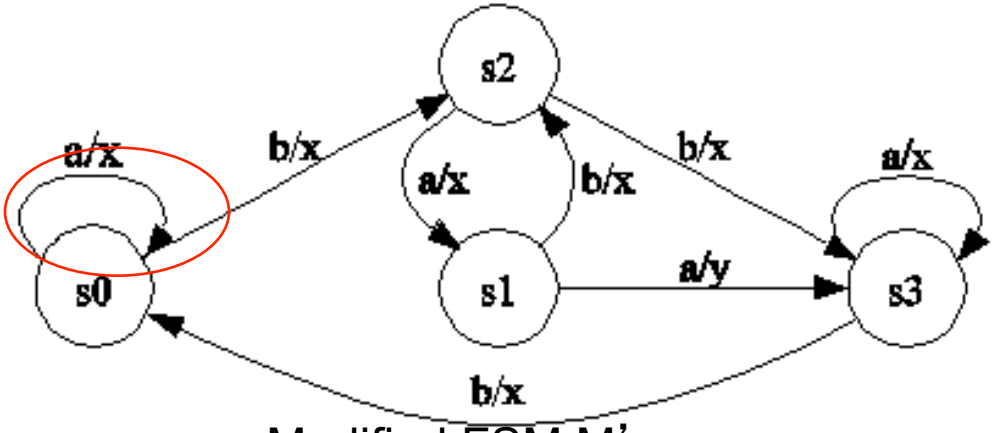



Eredeti FSM M

FSM M segédgráf $_{b}$




- **Változás:**
  - $\omega(s_0, a, x, s_1) = (s'_0, a, x, s'_0)$




Modified FSM M'

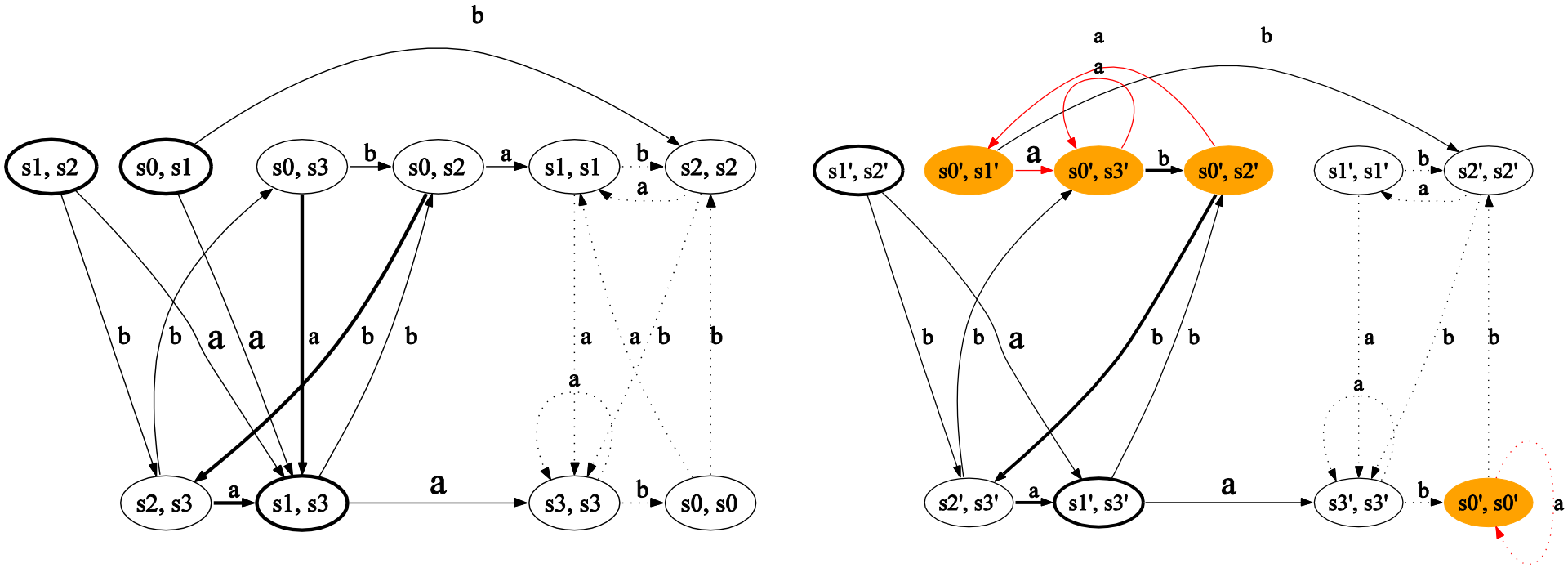
**Segédgráf:**

- **Nagyobb él címke:**
  - **"Separating input":**
  - **Az adott állapotpárt megkülönböztető input**
- **Vastagított ellipszis:**
  - **"Separating állapotpárok":**
  - **Állapotpárok separating inputtal**
- **Vastagított élek:**
  - **"Feszítő erdő":**
  - **Separating állapotpárba vezető utak**
- **Separating sequence:**
  - **A vastagított élek inputjai + a separating input**
- **Dotted edges:**
  - **Azonos állapotpárok közötti élek**
  - **Irrelevánsak**

# Példa: separating family of sequences 1/3

**1a) Változás végrehajtása**

- **Segédgráf módosítása**
- **Az a inputhoz tartozó él módosul (s0’,s0’), (s0’,s1’), (s0’,s2’), (s0’,s3’) esetén**




Auxiliary graph before and after the change

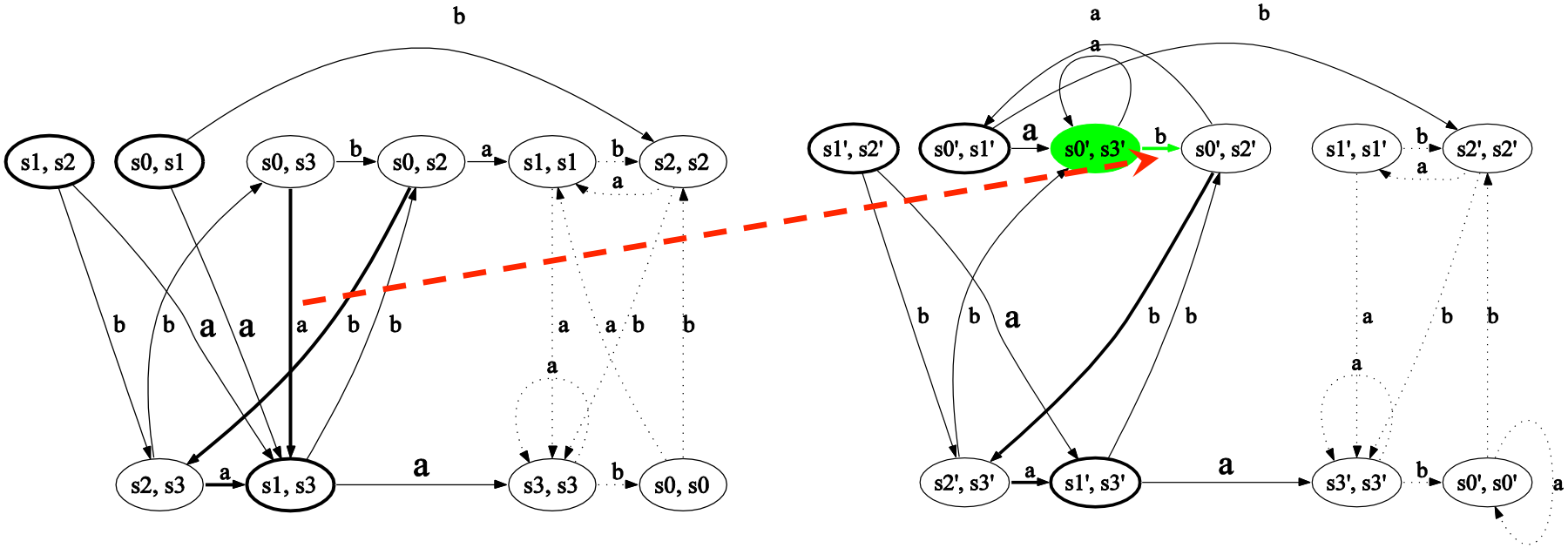
# Példa: separating family of sequences 2/3

**1b) Érintettek meghatározása**

- Egyedül (s0’,s3’) érintett

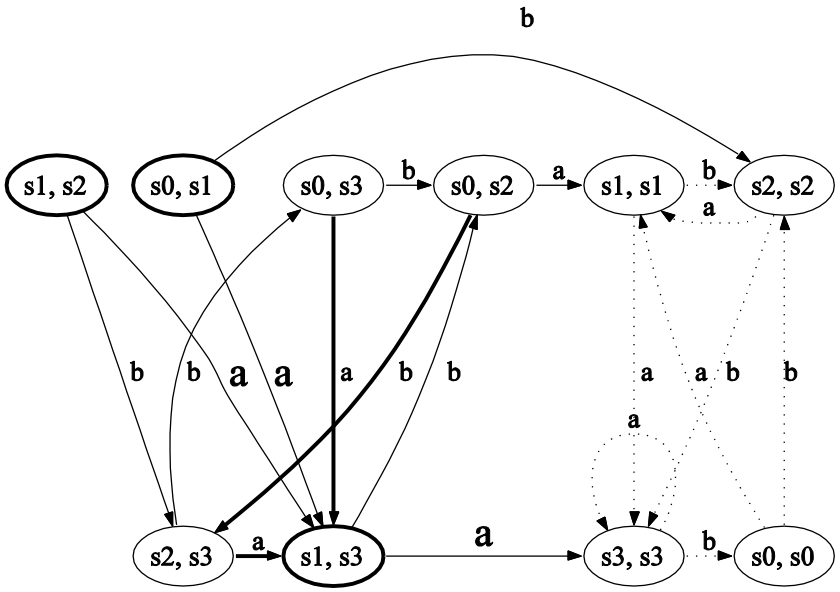
**2) Új elválasztó szekvencia létrehozása (s0’,s3’)-hoz**

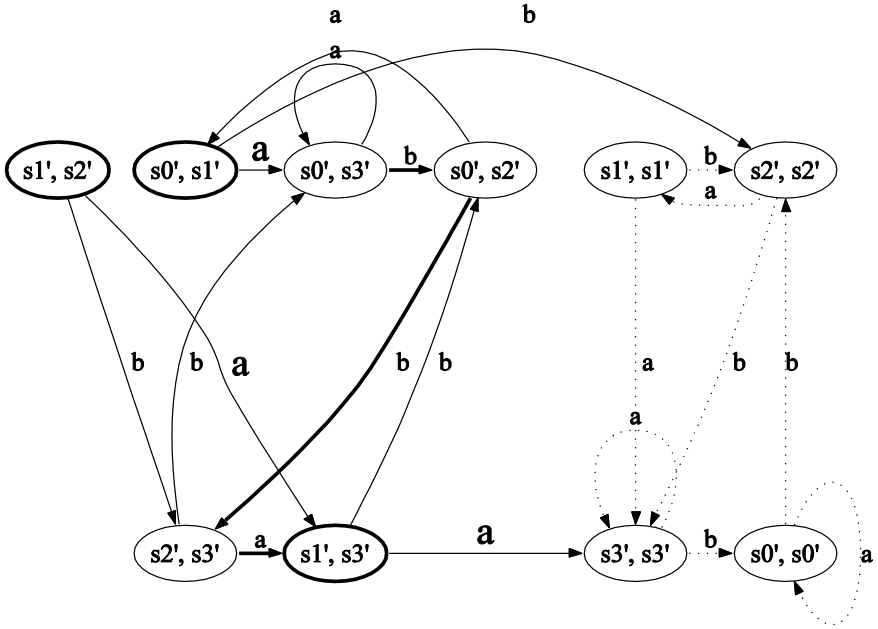
- b input esetén nem érintetthez jutunk (s0’,s3’)-ből



# Példa: separating family of sequences 3/3

- s0 és s3 separating szekvenciája változik
- a·a →b·b·a·a



# Komplexitás analízis

- A state cover set karbantartása:
  - $O(p*\Delta_Q)$, ahol $1 \leq \Delta_Q \leq n$
- A separating family of sequences karbantartása:
  - $O(p*\Delta_Z)$, ahol $n \leq \Delta_Z \leq n^2$
- Egy teszt szekvenciát frissíteni kell ha bármely része változik

- Inkrementális algoritmusokra általánosságban
  - Worst case esetben nem jobb mint a „batch” algoritmus
  - Ez triviális ha Δx jelentős x-hez viszonyítva
  - Viszont: Kis “atomic” változás is használhatatlanná tehet egy checking sequence-t
- Ez jellemző minden inkrementális algoritmusra
  - Valóságban mégis praktikus – worst case ritka