

# Modell alapú tesztelés

Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

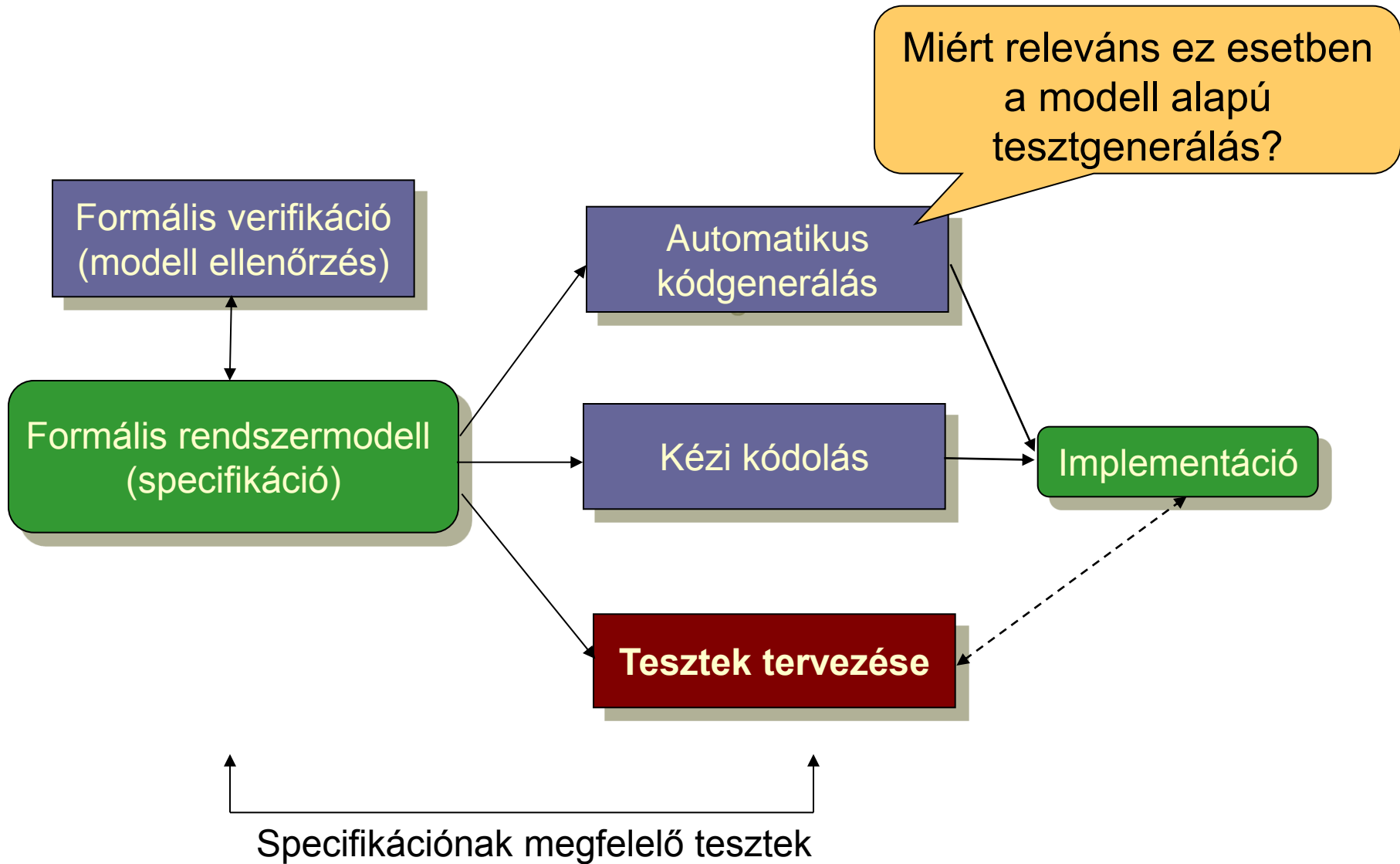
# Tartalomjegyzék

- Motiváció
  - Modellek (informális) szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- Tesztgenerálás fedettségi kritériumokhoz
  - Direkt algoritmusok
  - Modellellenőrzők használata
  - Tesztgenerálás korlátos modellellenőrzéssel
- Tesztgenerálás hibakészlet alapján
  - Modell mutációk
  - Ekvivalencia relációk tesztgeneráláshoz
- Eszközök a tesztgeneráláshoz

# Modell alapú tervezés (UML) és tesztelés

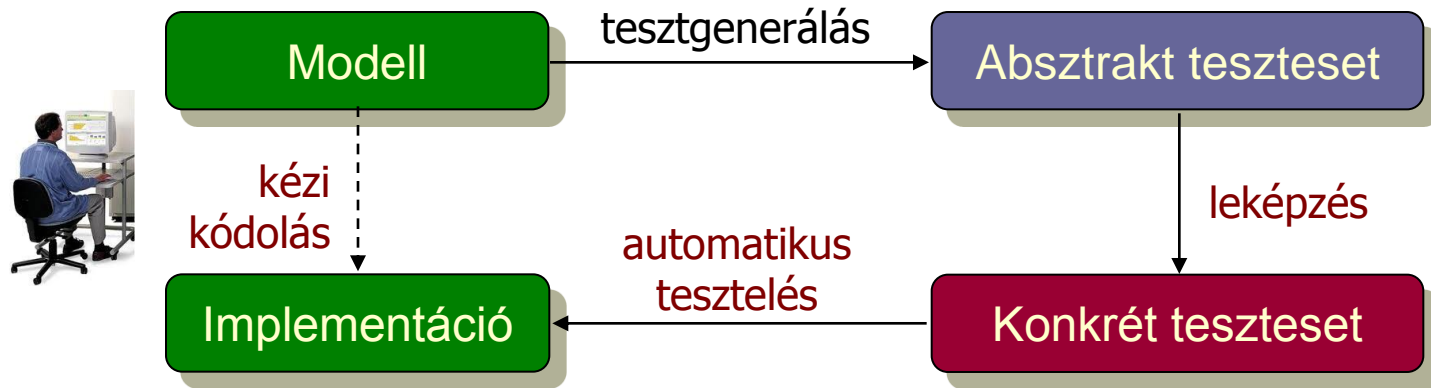
- Használati eset diagram:
  - Validációs tesztelés: tesztelendő használati esetek
- Osztály- és objektumdiagram
  - Modultesztelés: komponensek, interfészek azonosítása
- Állapottérkép és aktivitás diagram:
  - Modultesztelés: referencia struktúra alapú teszteléshez
- Üzenet-szekvencia és együttműködési diagram:
  - Integrációs tesztelés: forgatókönyvek származtatása
- Komponens diagram:
  - Rendszertesztelés: tesztelendő fizikai komponensek
- Telepítés diagram:
  - Rendszertesztelés: teszt konfiguráció

# Modell alapú fejlesztési folyamat (részlet)



# Használati esetek

- Kézi kódolás esetén: Konformancia ellenőrzés

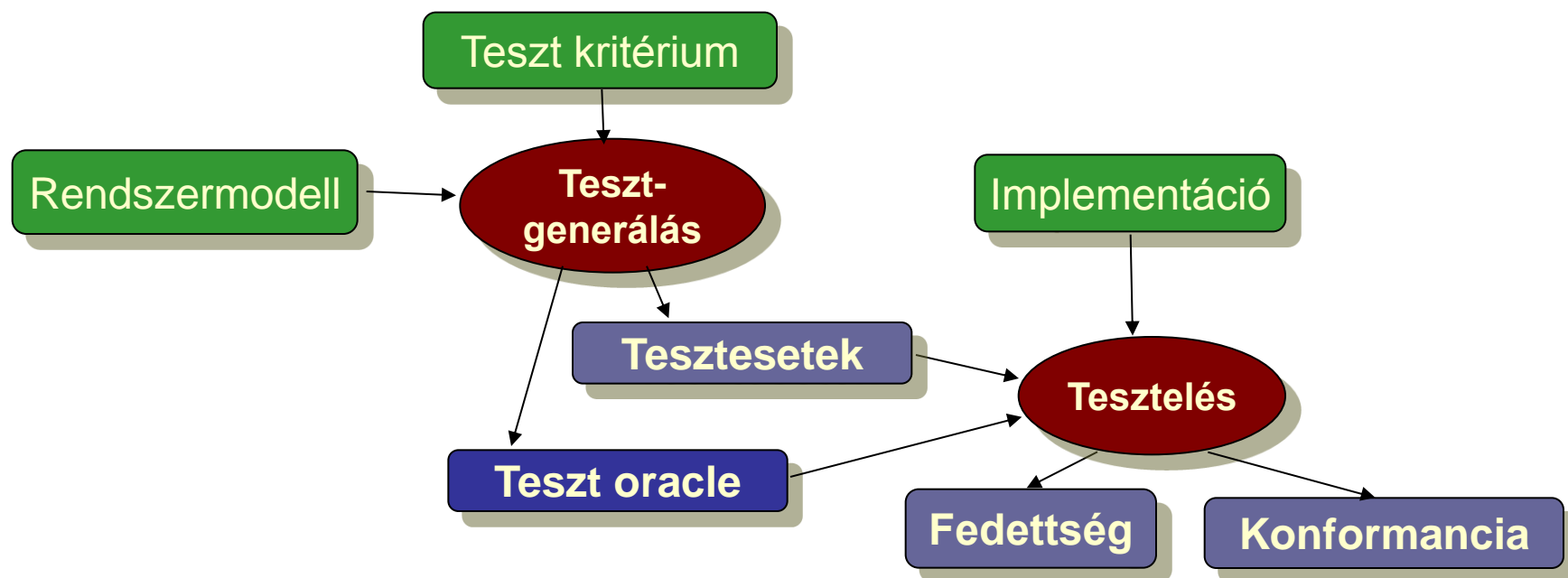


- Automatikusan kódgenerálás esetén: Validáció



# Modell alapú tesztelés alapfeladatai

- Rendszermodell és tesztelési kritérium alapján:
  - Tesztgenerálás (viselkedéshez, fedettséghez)
  - Teszt kiértékelő (test oracle) generálás
  - Teszt fedettség meghatározása
  - Konformancia megállapítása



# Tartalomjegyzék

- Motiváció
  - Modellek szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- **Tesztgenerálás fedettségi kritériumokhoz**
  - **Direkt algoritmusok**
  - Modellellenőrzők használata
  - Tesztgenerálás korlátos modellellenőrzéssel
- Tesztgenerálás hibakészlet alapján
  - Modell mutációk
  - Ekvivalencia relációk tesztgeneráláshoz
- Eszközök a tesztgeneráláshoz

# Tipikus alkalmazási terület

- **Állapot alapú, eseményvezérelt működés**
  - Eseményre triggerelt állapotátmenetek
  - Akciók (mint válasz jellegű kimenetek)
- **Felhasználói felületek tesztelése**
  - Eseményvezérelt működés
- **Egyszerű modellek használhatók**
  - Automaták (FSM; Mealy, Moore, Büchi, ...)
  - Magasabb szintű formalizmusok leképezhetők
    - UML állapottérkép
    - SCADE Safe Statechart
    - Simulink Stateflow
- **Gráfelméleti algoritmusok**
  - Algoritmus létezik sokféle tesztelési feladathoz
  - Optimális tesztek: Tipikusan NP-teljes algoritmusok ☹️



# Gráfelméleti algoritmus átmenet fedéshez

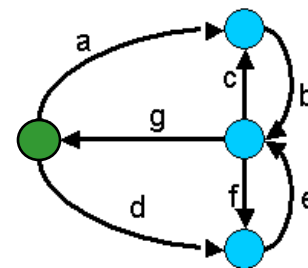
- Problémák megfeleltetése

- Tesztelési probléma: **Átmenetek fedése**

- Minden átmenet fedése teszt szekvenciával
- A teszt szekvencia vigyen vissza a kezdeti állapotba

- Gráfelméleti probléma: **„New York-i utcaseprő” probléma**

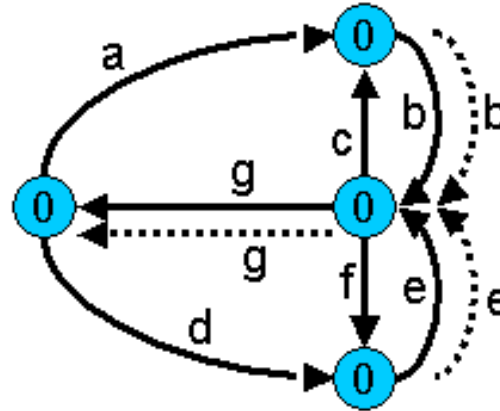
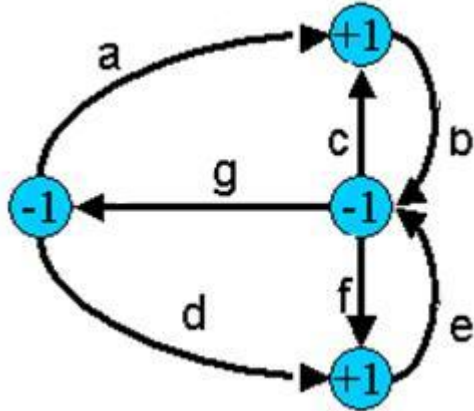
- Egy irányított gráfban mi az a (legrövidebb) bejárási szekvencia, ami minden élet bejár és a kezdeti helyre visz vissza?
- (Ugyanez nem irányított gráfban: „Kínai postás” probléma)



- Megoldás alapötlete:

- Helyek polaritásainak számítása: Bejövő mínusz kimenő élek száma
- Olyan élek duplikálása, amelyek pozitívtól negatív polaritású helyekig vezetnek, amíg minden hely nulla polaritású nem lesz
- Euler-kör keresése az így adódó gráfban (lineáris algoritmus)
  - Euler-kör: Minden élet bejár; ilyen gráfban biztosan képezhető
- Az Euler-kör bejárása adja a teszt szekvenciát

# Egy példa átmenet fedéshez



Eredeti gráf  
hely polaritásokkal

Duplikált élekkel  
kiegészített gráf (Euler-gráf)

Bejárási szekvencia (Euler-kör):

a b c b f e g d e g

# Gráfelméleti algoritmus átmenet kombináció fedéshez

- Problémák megfeleltetése

- Tesztelési probléma: **Átmenet kombinációk fedése**

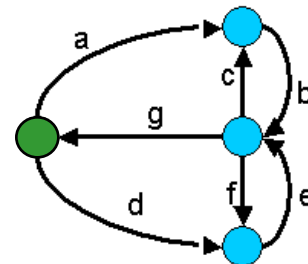
- Minden lehetséges, egymás után  $n$  számú átmenetből álló sorozat fedése a teszt szekvenciával
    - A teszt szekvencia vigyen vissza a kezdeti állapotba
    - Legegyszerűbb eset: Minden lehetséges **átmenet-pár** fedése

- Gráfelméleti probléma: **„Bankrabló” probléma**

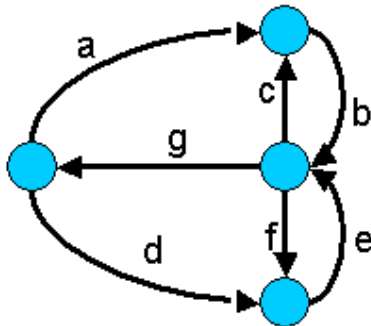
- (Legrövidebb) élszekvencia, amiben minden lehetséges  $n$  hosszú élsorozat előfordul (legegyszerűbb eset:  $n=2$ )

- Megoldás (de Bruijn algoritmus) alapötlete ( $n=2$ ):

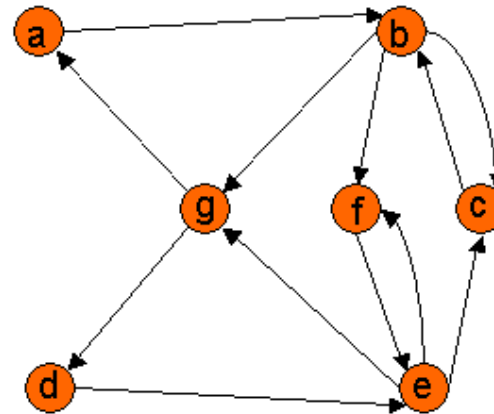
- Duális gráf megkezdése: Az eredeti gráf éleiből helyek lesznek
  - Az eredeti gráfban létező élpárok esetén él behúzása a duális gráfba az élek által adott helyek közé
  - A duális gráf kiegészítése (élek duplikálásával) Euler-gráffá
  - Az így kapott gráfban az Euler-kör adja a teszt szekvenciát



# Egy példa átmenet kombináció fedéshez



Eredeti gráf



Duális gráf az élpárokkal

Eredeti bejárás szekvencia az élek fedéséhez:

a b c b f e g d e g

PI. a b, g élpár nincs lefedve!

Bejárás szekvencia a duális gráf alapján élpárok fedéséhez:

a b c b f e c b g d e f e g

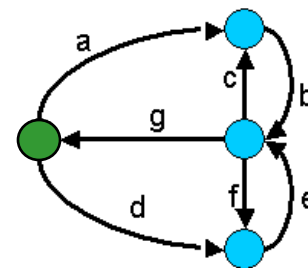
# Gráfelméleti algoritmus konkurens átmenet fedéshez

- Problémák megfeleltetése

- Tesztelési probléma:

- Konkurens tesztelés átmenetek fedéséhez

- Teljes átmenet fedés a cél, de több tesztelő van
      - Célszerű egyenletesen megosztani a problémát, hogy a legrövidebb idő alatt végezzenek; mindegyik a kezdőállapotból kezd
      - Feltétel: Egy bemenettel bárhonnán kezdőállapotba vihető a rendszer

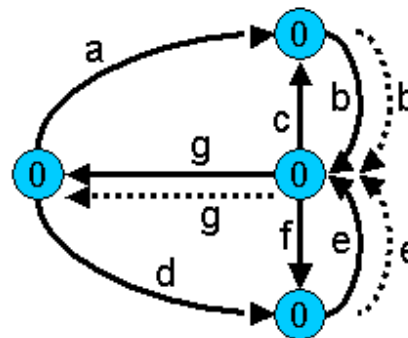
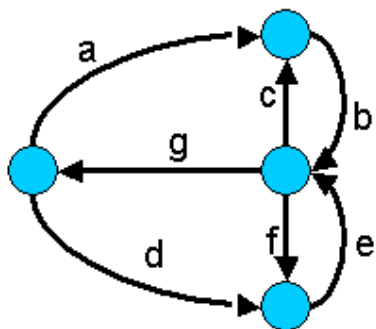


- Gráfelméleti probléma: „Utcapró brigád” probléma

- Megoldás: Egy heurisztika (nem optimális megoldás)

- Egy-egy bejáráshoz **k** felső határ megadása
  - Olyan élszekvencia keresése, amely a legtöbb eddig nem érintett élet tartalmazza, de legfeljebb **k** hosszú; a végén kezdőállapotba vezérelve a bejárást
  - Ezután újabb élszekvenciák felvétele, amíg van be nem járt él
  - A **k** felső határral lehet próbálkozni

## Egy példa konkurens átmenet fedéshez



Eredeti bejárási szekvencia (Euler-kör):

a b c b f e g d e g

Egy lehetséges (nem optimális) megosztás:

- Tesztelő 1: a b c b f e g (7 időegység kell)
- Tesztelő 2: d e g

Egy jobb megosztás (heurisztikával):

- Tesztelő 1: a b c b g (5 időegység kell)
- Tesztelő 2: d e f e g

# Tartalomjegyzék

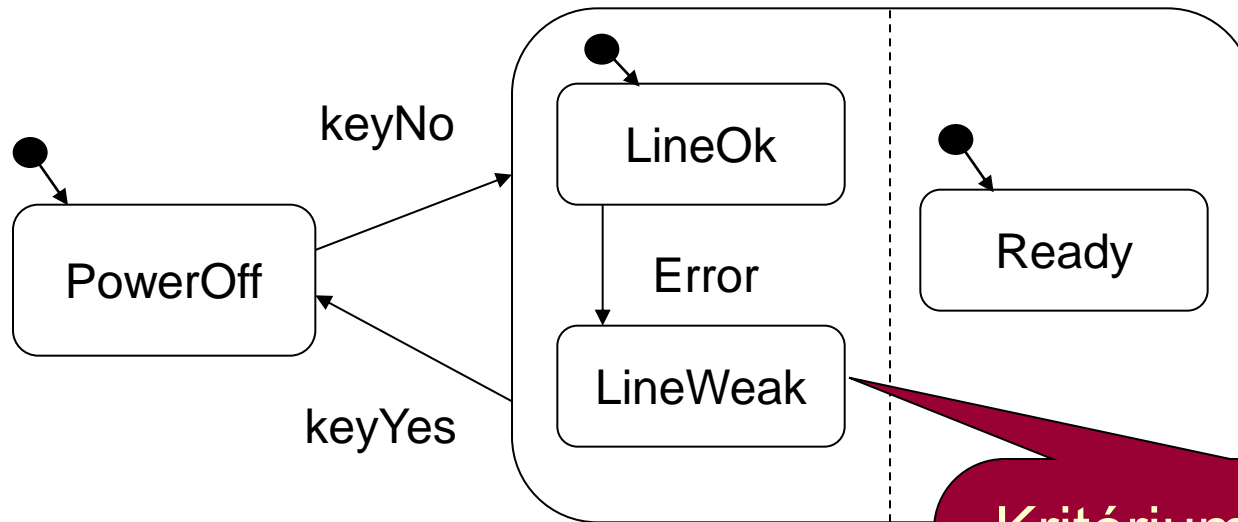
- Motiváció
  - Modellek szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- **Tesztgenerálás fedettségi kritériumokhoz**
  - Direkt algoritmusok
  - **Modellellenőrzők használata**
  - Tesztgenerálás korlátos modellellenőrzéssel
- Tesztgenerálás hibakészlet alapján
  - Modell mutációk
  - Ekvivalencia relációk tesztgeneráláshoz
- Eszközök a tesztgeneráláshoz

# Alapötlet

- Tipikus tesztelési kritériumok:
  - Állapotok, átmenetek lefedése
  - Változó definiálások és felhasználások lefedése
  - Be- és kimenő átmenet-párok lefedése egy-egy állapothoz
- Tesztgeneráláshoz szükséges:
  - Állapottér bejárása
    - Modellellenőrző is ezt csinálja
- Alapötlet:
  - Járja be a modellellenőrző az állapotteret!
  - Irányítsuk úgy, hogy az általa adott **ellenpélda** legyen a teszteset!

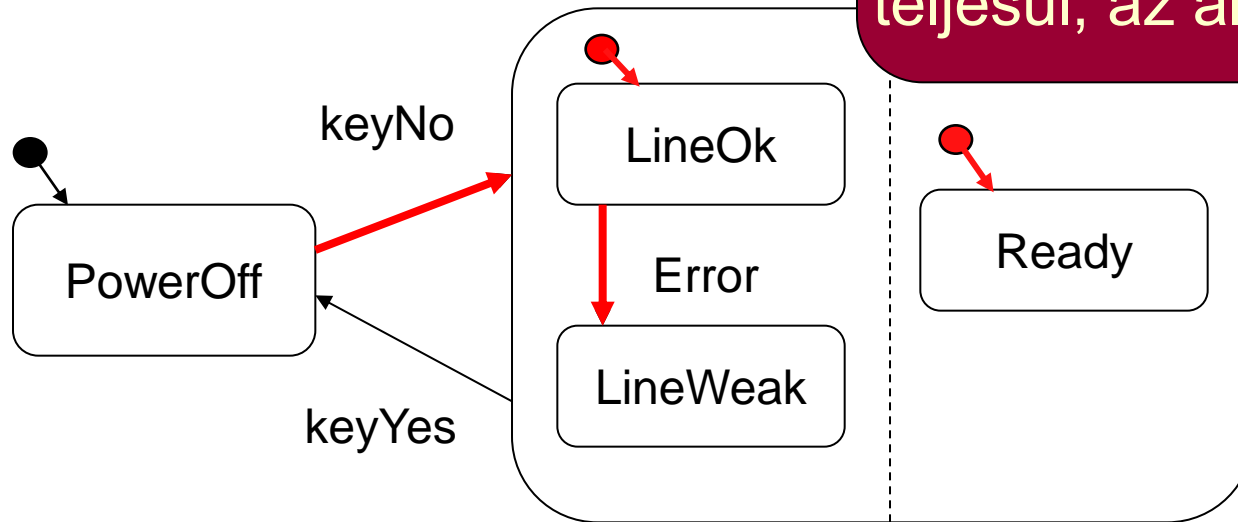


# A modellellenőrző használata tesztgenerálásra



Kritérium megadása:  
A LineWeak állapotot  
soha sem lehet elérni:  
 $\neg$  (EF LineWeak)

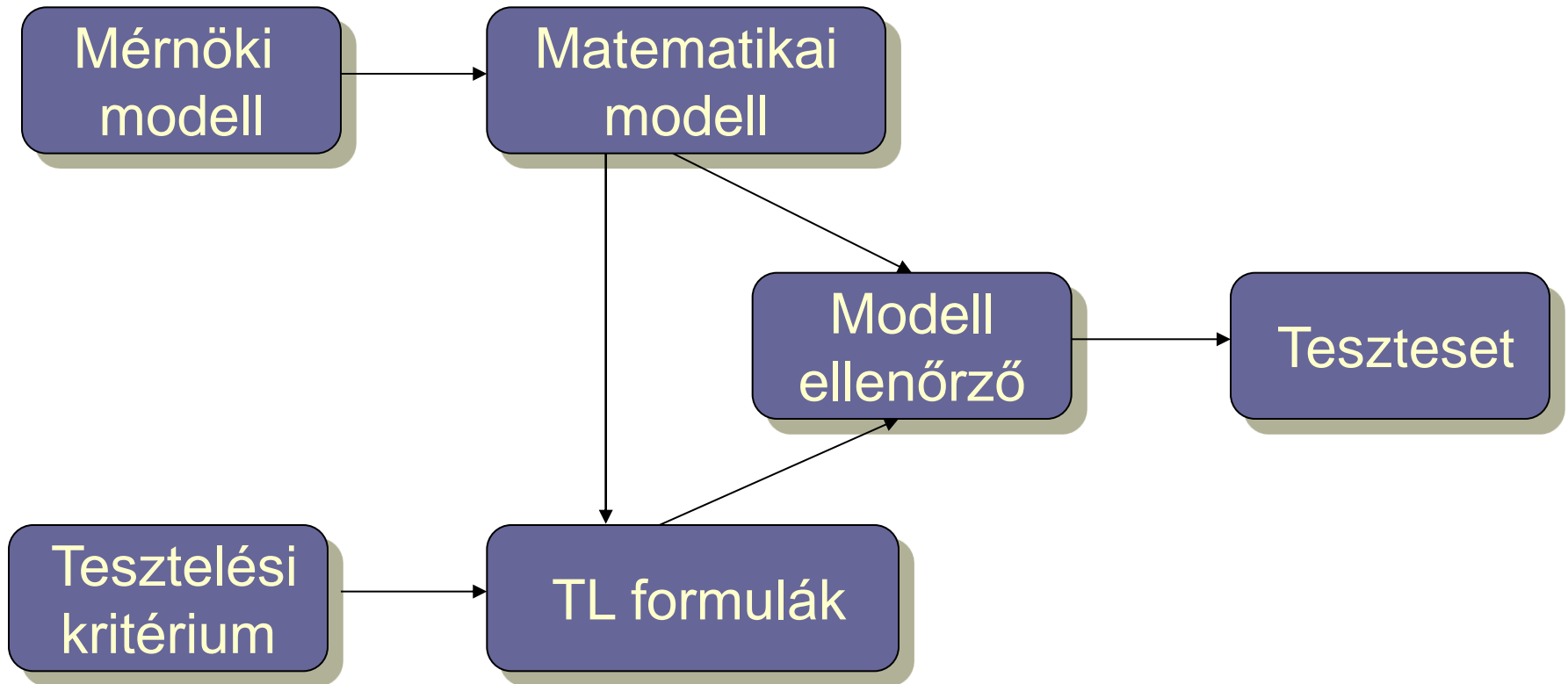
# A modellellenőrző használata tesztergenerálásra



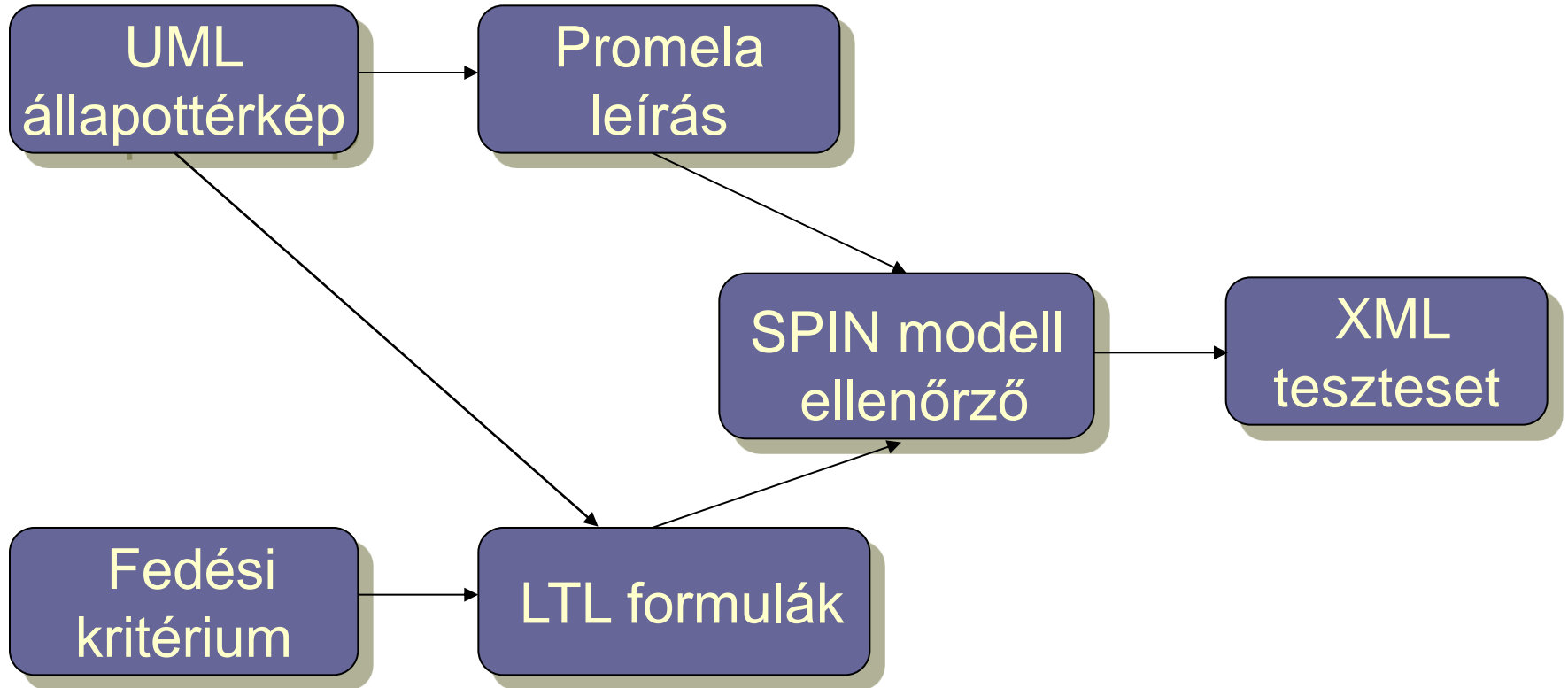
A modellellenőrző ezzel az ellenpéldával demonstrálja, hogy a tulajdonság nem teljesül, az állapot elérhető.

Ez viszont pontosan egy, a LineWeak állapotot lefedő teszteset!

# Automatikus tesztergenerálás



# Egy megvalósítás



# Fedettségi kritériumok mint TL kifejezések

- Címkék a modellben  $v$  változóra (predikátumok):
  - $\text{def}(v)$
  - $\text{c-use}(v)$
  - $\text{p-use}(v)$
  - $\text{implicit-use}(v)$
- Karakterisztikus függvények (állapotváltozókkal):
  - $s$ : adott  $s$  állapotban való tartózkodás
  - $t$ : adott  $t$  átmenet tüzelése (állapot és következő állapot)
- Állapothalmazok ( $\rightarrow$  predikátumok diszjunkcióval):
  - $d(v)$ : minden  $\text{def}(v)$
  - $u(v)$ : minden  $\text{c-use}(v)$  vagy  $\text{p-use}(v)$
  - $\text{im-u}(v)$ : minden  $\text{implicit-use}(v)$
  - $\text{exit}$ : megfelelő állapotok új teszthez (pl. kezdőállapotok)

A változó használata implicit átmenet feltételében.  
Implicit átmenet: Helyben maradást jelent (az adott feltétel mellett); ez is tesztelhető.

# Vezérlés alapú fedettségi kritériumok

- Állapotfedés:

$$\{\neg EF s \mid s \text{ alapszintű állapot}\}$$

Kritériumhalmaz!

Ha megfelelő stabil állapot is kell újabb teszthez:

$$\{\neg EF (s \wedge EF \text{ exit}) \mid s \text{ alapszintű állapot}\}$$

(a további képletekben EF exit kihagyva)

- Gyenge átmenet fedés:

$$\{\neg EF t \mid t \text{ átmenet}\}$$

- Erős átmenet fedés:

$$\{\neg EF t \mid t \text{ átmenet}\} \cup \{\neg EF it \mid it \text{ implicit átmenet}\}$$

Erős fedés: Implicit átmenetek (helyben maradás) is tesztelve

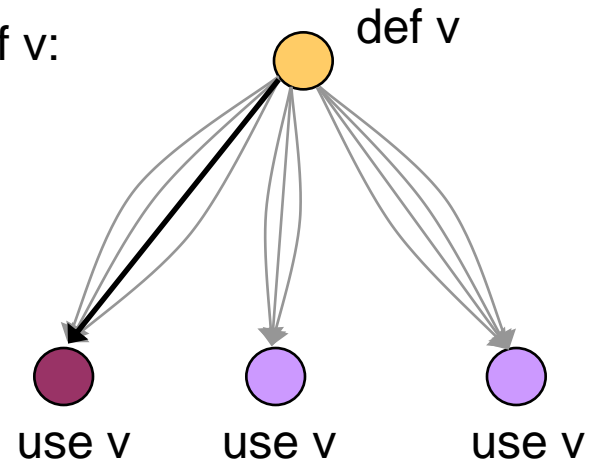
# Adatfolyam alapú fedettségi kritériumok (ismétlés)

- All-defs:

minden  $v$ , minden def  $v$ :

egy def-clear  
útvonal:

**egy** use  $v$ :

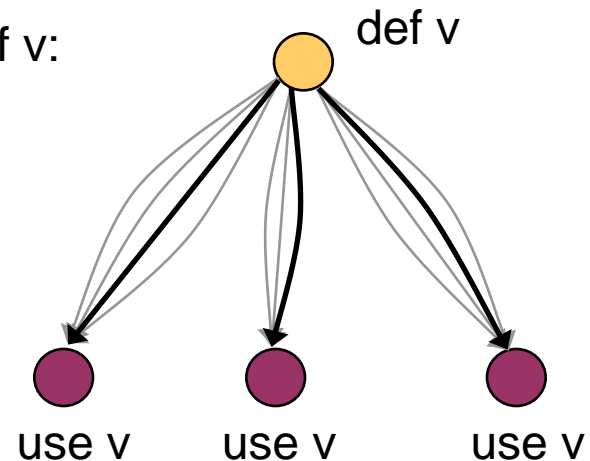


- All-uses:

minden  $v$ , minden def  $v$ :

egy def-clear  
útvonal:

**minden** use  $v$ :



# Adatfolyam alapú fedettségi kritériumok

- Gyenge all-defs fedés:

Egy def-clear útvonal tesztelve minden def(v) és egy use(v) között

$$\{\neg EF (t \wedge EX E(\neg d(v) \cup u(v))) \mid v \text{ változó, } t \in d(v)\}$$

- Gyenge all-uses fedés:

Egy def-clear útvonal tesztelve minden def(v) és minden use(v) között

$$\{\neg EF (t \wedge EX E(\neg d(v) \cup t')) \mid v \text{ változó, } t \in d(v), t' \in u(v)\}$$

- Erős all-defs fedés:

Implicit változó használat (~ helyben maradás feltétele) is tesztelve

$$\{\neg EF (t \wedge EX E(\neg d(v) \cup (u(v) \vee \text{im-}u(v)))) \mid v \text{ változó, } t \in d(v)\}$$

- Erős all-uses fedés:

$$\{\neg EF (t \wedge EX E(\neg d(v) \cup t')) \mid v \text{ változó, } t \in d(v), t' \in u(v) \cup \text{im-}u(v)\}$$



# Korlátozások

- Modellellenőrző jellegzetességei:
  - Csak egy ellenpéldát generál
  - Így nem generálhatók tesztek olyan fedettségi kritériumokhoz, ahol minden ellenpéldára szükség van
    - Pl. **all-du-paths** kritérium  
(minden def-clear útvonal egy def-use párhoz)
- Absztrakt teszt eset adódik
  - Csak a bemeneti szekvencia kötött
  - Elvárt kimeneteket meg kell határozni (szimulációval)
- Nemdeterminisztikus modellek:
  - Egy bemeneti szekvenciához több bejárás (cél állapot)
  - Teszt végrehajtásakor figyelembe kell venni

# Optimalizáció

- Modellellenőrző feladata:
  - Állapottér hatékony bejárása: Gyorsan, kis tárigénnyel
- A tesztgenerálás célja:  
Gyorsan minél rövidebb ellenpéldát találni
  - Speciális beállítások szükségesek a modellellenőrzőben
  - Legrövidebb/legkisebb tesztkészlet kiválasztása:  
NP-teljes probléma!
- Lehetőségek (pl. SPIN esetén):
  - Szélességi keresés az állapottérben (BFS)
  - Mélységi keresés, de mélységkorláttal (limited DFS)
  - Rövidebb ellenpélda iteratív keresése
  - Közelítő modell ellenőrzés (hash fv. az állapottároláshoz)
    - Bizonyos állapotokat nem jár be a keresés során
    - De ha talál ellenpéldát, az valós teszt lesz

# Tesztgenerálási eredmények

Options (compile or run-time)	Time required for test generation	Length of the test sequences	Longest test sequence
-i	22m 32.46s	17	3
-dBFS	11m 48.83s	17	3
-i -m1000	4m 47.23s	17	3
-l	2m 48.78s	25	6
default	2m 04.86s	385	94
-l -m1000	1m 46.64s	22	4
-m1000	1m 25.48s	97	16
-m200 -w24	46.7s	17	3

## Paraméterek:

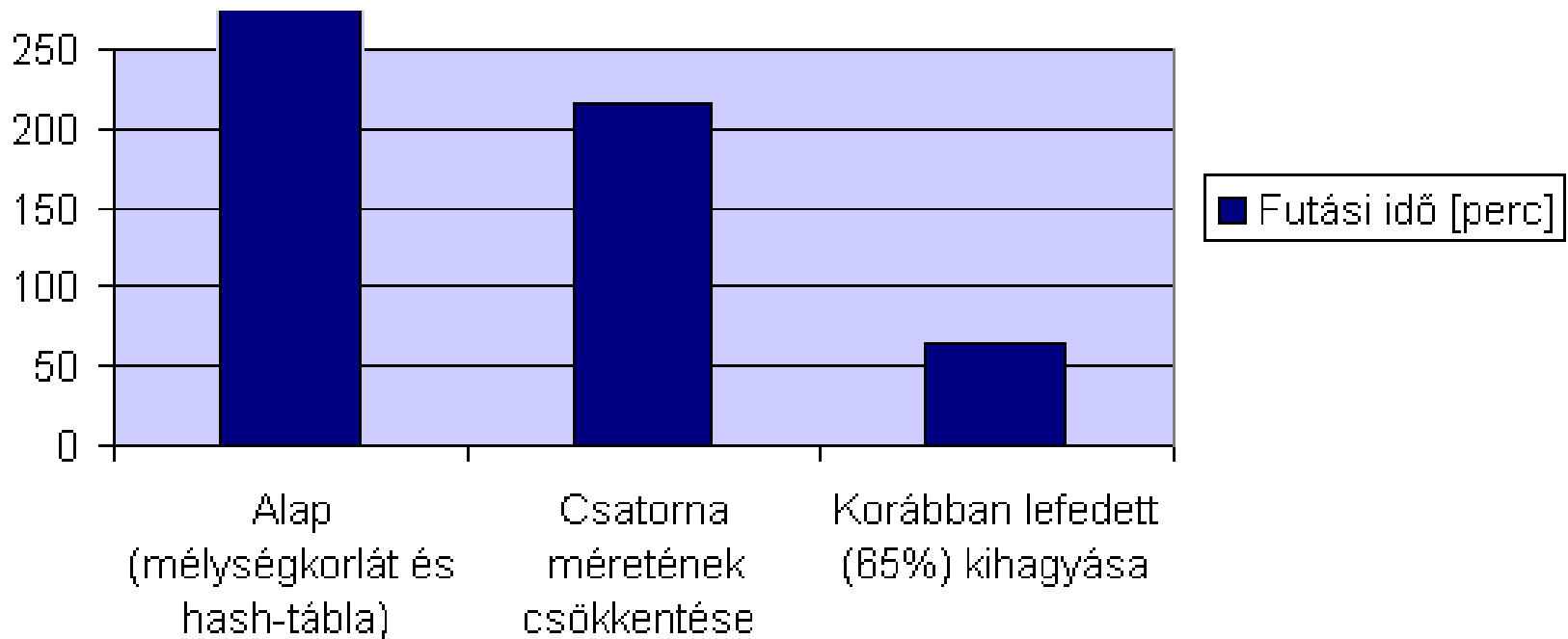
- -i iteratív, -l közelítő iteratív
- -dBFS szélességi keresés
- -m mélységi keresés korlátja
- -w hash tábla korlátja

Mobiltelefon viselkedését leíró  
állapotgép (10 állapot, 11  
átmenet)

# „Bitszinkronizációs protokoll” példa

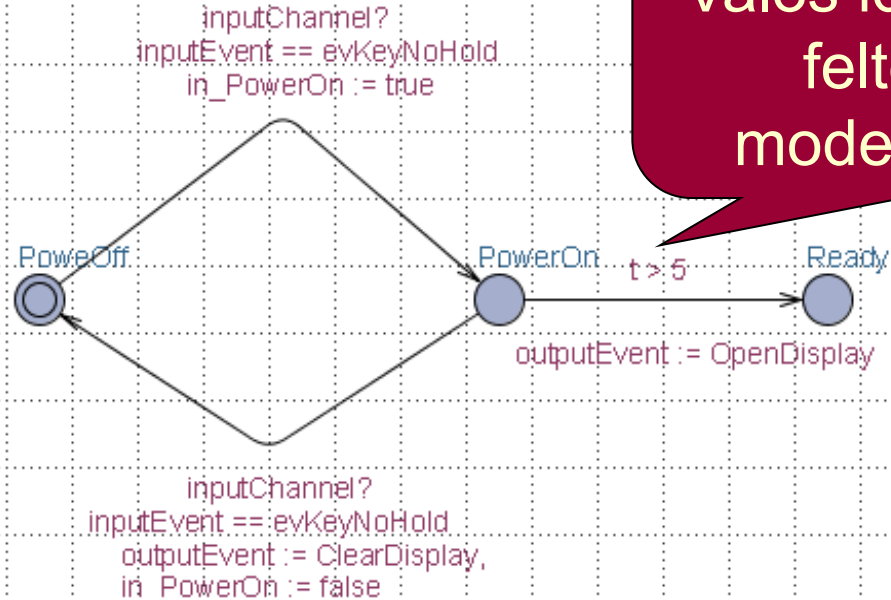
- Példa: Bitek szinkronizálása egy elosztott rendszerben
  - 5 objektum, 31 állapot, 174 átmenet
  - 2e+08 bejárando állapot
- Más technikák is kellenek:
  - Erősen tömörítő állapottárolás alkalmazása (bitstate hashing)
  - Szűkítések a modellben: csatorna méret csökkentés
  - Korábban lefedett kritériumok kihagyása
- További heurisztikák alkalmazása:
  - Mélyen fekvő állapotok tesztelése előbb

# „Bitszinkronizációs protokoll”: Tesztek generálása teljes állapotfedéshez



# Kiterjesztés valós idejű rendszerekre

Óra változók:  
Valós időt, időzítési  
feltételeket  
modellezhetünk



Időzített automaták használata

Speciális modell ellenőrző: UPPAAL

# Generált tesztek

State:

( input.sending mobile.PowerOn mobile1.LineOK mobile2.CallWait )

t=0 inputEvent=28 outputEvent=14

**Delay: 6**

State:

( input.sending mobile.PowerOn mobile1.LineOK mobile2.CallWait )

t=6 inputEvent=28 outputEvent=14

Transitions:

input.sending->input.sendInput { 1, inputChannel!, 1 }

mobile2.CallWait->mobile2.VoiceMail { inputEvent == evKeyYes && t > 5 &&  
in\_PowerOn, inputChannel?, 1 }

A teszt időzítési viszonyok is szerepelnek a generált tesztesetben

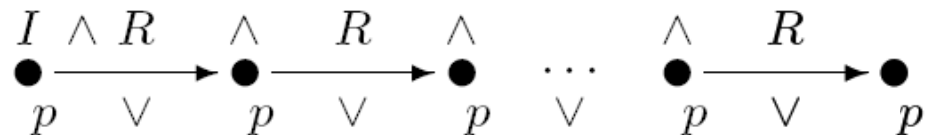
# Tartalomjegyzék

- Motiváció
  - Modellek szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- **Tesztgenerálás fedettségi kritériumokhoz**
  - Direkt algoritmusok
  - Modellellenőrzők használata
  - **Tesztgenerálás korlátos modellellenőrzéssel**
- Tesztgenerálás hibakészlet alapján
  - Modell mutációk
  - Ekvivalencia relációk tesztgeneráláshoz
- Eszközök a tesztgeneráláshoz

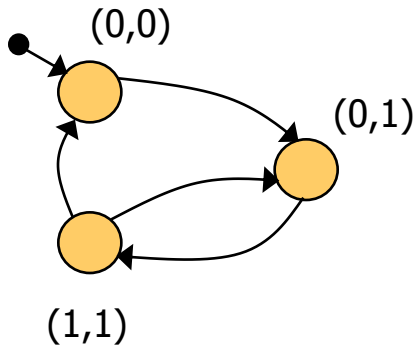


# Alapötlet: Korlátos modellellenőrzés alkalmazása

- SAT probléma megoldóinak használata
  - SAT megoldó: Boole függvényekhez keres helyettesítési értéket, ami a függvény értékét igazá teszi
- A modell elemeinek leképezése logikai függvénybe:
  - Kezdőállapotokra vonatkozó predikátum:  $I(s)$
  - Elérendő állapotokra vonatkozó predikátum:  $p(s)$
  - Állapotátmeneti reláció:  $R(s, s')$ 
    - Lépésenkénti „széthajtogatás”:  $R(s_i, s_{i+1})$
- A logikai függvény felírása: Konjunkció
  - Kezdőállapotból indul: Az  $I(s)$  predikátum az első állapotra
  - Széthajtogatott átmenetek: Az  $R(s_i, s_{i+1})$  reláció alkalmazása
  - Elérendő állapot: A  $p(s)$  predikátum valahol fennáll



# Példa: A modell leképzése logikai függvénybe

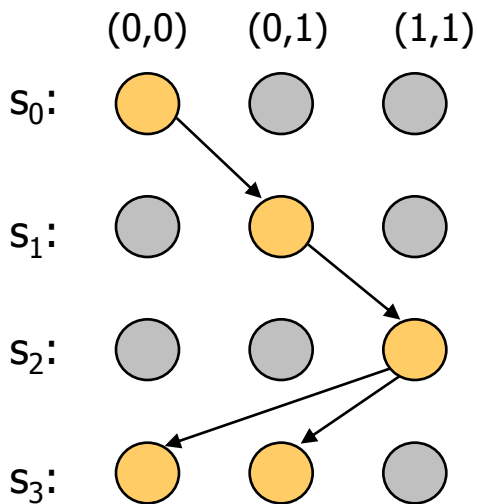


Kezdőállapot predikátum:

$$I(x,y) = (\neg x \wedge \neg y)$$

Állapotátmeneti reláció:

$$\begin{aligned}
 R(x,y,x',y') = & (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee \\
 & \vee (\neg x \wedge y \wedge x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge \neg y')
 \end{aligned}$$



3 lépéses kihajtogatás a kezdőállapotból:

$$\begin{aligned}
 & I(x_0, y_0) \wedge \\
 & R(x_0, y_0, x_1, y_1) \wedge \\
 & R(x_1, y_1, x_2, y_2) \wedge \\
 & R(x_2, y_2, x_3, y_3)
 \end{aligned}$$

# SAT alapú tesztgenerálás fedési kritériumokhoz

- Formula konstruálás:
  - Kihajtogatás  $k$  lépésben a kezdőállapotból
  - Teszt kritérium megadása: **TG** formula, pl.:
    - Adott állapot elérése
    - Adott állapotátmenet végrehajtása
    - Adott modellrészlet bejárása, ...

$$\exists s_0, s_1, \dots, s_k : I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge TG$$

The diagram illustrates the decomposition of the SAT formula into three components. The formula is written as  $\exists s_0, s_1, \dots, s_k : I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge TG$ . Below the formula, three downward-pointing chevrons connect the terms to their respective labels:  $\exists s_0, s_1, \dots, s_k$  is labeled "Állapot-szekvencia",  $I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$  is labeled "Modell széthajtogatás", and  $TG$  is labeled "Teszt cél".

- Ha ez a formula **kielégíthető**, akkor az egy tesztet ad:
  - A teszt teljesíti a TG kritériumot
  - Ha nem kielégíthető a formula, akkor nincs teszt a kritériumhoz

# Használhatóság

- A tesztgenerálás korlátai
  - Legfeljebb adott hosszúságú teszt generálható
    - Iteratíván növelhető a kihajtogatás korlátja
  - Így részleges megoldás adódik
    - Amit megtalál, az biztosan teszt eset lesz
    - Nem garantált, hogy megtalálja a teszt esetet (ha az hosszabb lenne, mint amit figyelembe veszünk)
- A modellből SAT probléma leképzése automatikus
- A TG teszt célok megadása egyszerűsíthető
  - C programokhoz: FQL nyelv teszt célokhoz (FSHELL)  
`in /code.c/ cover @line(6),@call(f1) passing @file(code.c) \ @call(f2)`
  - Elő- és utófeltételek megadása:
    - Van-e olyan teszt eset, amikor az utófeltétel nem teljesül?
    - Ellenpélda generálás

# Tartalomjegyzék

- Motiváció
  - Modellek szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- Tesztgenerálás fedettségi kritériumokhoz
  - Direkt algoritmusok
  - Modellellenőrzők használata
  - Tesztgenerálás korlátos modellellenőrzéssel
- **Tesztgenerálás hibakészlet alapján**
  - **Modell mutációk**
  - **Ekvivalencia relációk tesztgeneráláshoz**
- Eszközök a tesztgeneráláshoz

# Hibakészletek használata

- Tapasztalatok a szoftver tesztelés során
  - **Csatolási effektus** (coupling effect): Azok a teszt esetek, amik egyszerű hibákat megtalálnak, bonyolultabbakra is hatékonyak
  - **Kompetens programozó hipotézis**: A programok általában jók, a hibák nagy része gyakran előforduló tipikus hiba
- Alapötlet:
  - Állítsunk elő olyan „mutáns” modelleket, amik **tipikus hibákat** tartalmaznak, és generáljunk ezek kimutatására tesztek
  - Ezek várhatóan **bonyolultabb hibákhoz** is jobbak a véletlen tesztekénél
- Tipikus mutációk:
  - Aritmetikai operátorok felcserélése feltételekben
  - Akciók (műveletek, üzenetek) sorrendjének megváltoztatása
  - Akciók kihagyása
  - ...

# Tesztgenerálás hibakészlet alapján

- A tesztgenerálási feladat:
  - Olyan tesztek előállítása, amelyek **különbséget tesznek** az eredeti (hibamentes) és a mutáns (hibás) viselkedés között
  - Ezek ún. negatív tesztek (sikertelen teszt: nincs hiba!)
- Hogyan definiáljuk a „különbséget” két viselkedés között?  
Milyen különbség megengedett?
  - Más viselkedés megengedett-e a specifikált mellett?
    - Több kimenet, más bemenetekre való reakció, ...
  - Kihagyás (elmaradt kimenet) megengedett-e?
- Szokásos megoldások
  - Biztonságkritikus rendszer:
    - Szigorúan a specifikáció szerint
    - Teljes specifikáció szükséges
  - „Hétköznapi” rendszer (akár fejlesztés közben):
    - Beférjen a specifikáció keretei közé

# k-ekvivalencia a teszteléshez

- Alkalmazás: Fekete doboz teszteléshez
  - Bemenetek egy  $s$  állapotban:  $in(s)$  – vezérelhetők
  - Kimenetek egy  $s$  állapotban:  $out(s)$  – megfigyelhetők
  - Kimeneti akció hiánya is formalizálható: Speciális  $\delta$  akció
- A k-ekvivalencia definíciója:

Azonos bemeneti sorozat mellett azonos kimenetek az első  $k$  lépésre

- Jelölések:

Kezdeti állapot predikátum:

Állapotátmeneti reláció:

Eredeti  $M$  modell:

$I(s_0)$

$R(s_i, s_{i+1})$

Mutáns  $M'$  modell:

$I'(s'_0)$

$R'(s'_i, s'_{i+1})$

A modell kihajtogatása  $k$  lépésre:  $I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$



# Mutáció alapú tesztgenerálás k-ekvivalencia alapján

- SAT formula konstruálás a k-ekvivalenciához:
  - Azonos bemeneti szekvencia mindkét modellre
  - Kihajtogatás k lépésben az eredeti modellre
  - Kihajtogatás k lépésben a mutáns modellre
  - Legalább egy különböző kimenet lesz a kimeneti szekvenciában

$$\bigwedge_{i=0}^k (\text{in}(s_i) = \text{in}(s'_i)) \wedge I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge I'(s'_0) \wedge \bigwedge_{i=0}^{k-1} R'(s'_i, s'_{i+1}) \wedge \bigvee_{i=0}^k (\text{out}(s_i) \neq \text{out}(s'_i))$$

Egyező  
bemenetek

Eredeti  
modell

Mutáns  
modell

Eltérő  
kimenet

- Ha ez a formula kielégíthető, akkor az egy tesztet ad
  - A teszt különbséget tesz a modellek között:  
Kimutatja a mutációt, tehát a hiba felderítésére használható
  - Ha a formula nem kielégíthető, akkor ekvivalens a két modell

# Mutáció alapú tesztgenerálás az IOCO reláció alapján

- Az IOCO reláció informálisan:
  - Megengedett, hogy azonos bemeneti szekvenciára a **mutáns modell** kimenetei **részalmazát** képezik az eredeti modellben rögzített kimeneteknek (azaz „beleférnek” az eredeti modellbe)
    - Részleges viselkedés (kihagyás) megengedett, eltérő viselkedés nem
    - Többlet funkció megengedett az eredeti modellben nem rögzített bemeneti szekvenciára
  - A k-ekvivalenciánál megengedőbb konformancia reláció
- Definíció (ld. korábban):

Minden, az eredeti modellben felvehető akciószekvenciára igaz:  
Az így elérhető **állapotokban** a **mutáns** által nyújtott **kimeneti akciók** részalmazát képezik az **eredeti modell** által nyújtott **kimeneti akcióknak**
- Tesztek generálhatók SAT megoldóval
  - Bonyolultabb a részalmaz reláció vizsgálata miatt (itt nem írjuk fel)

# IOCO alapú tesztgenerálás jellegzetességei

- **Teszt eset:**
  - Egy olyan akciószekvencia, ami különbséget tesz az eredeti és a mutáns modell között: kimutatja, hogy nem IOCO ekvivalensek
  - A SAT megoldó miatt korlátos hosszúságú teszt eset
- **Jellegzetességek:**
  - Tartalmazza bemeneti és a kimeneti sorozatot is (nem kell utólagos bejárás)
  - Nemdeterminisztikus modell esetén csak teszt célként fogható fel
  - A  $\delta$  akció detektálása a teszt végrehajtás során mint **timeout** jelenik meg
    - A timeout elfogadható a specifikációban  $\delta$  -val jelölt állapotokban (ez az elvárt viselkedés)

# Tartalomjegyzék

- Motiváció
  - Modellek szerepe a tesztelésben
  - Modell alapú tesztgenerálás
- Tesztgenerálás fedettségi kritériumokhoz
  - Direkt algoritmusok
  - Modellellenőrzők használata
  - Tesztgenerálás korlátos modellellenőrzéssel
- Tesztgenerálás hibakészlet alapján
  - Modell mutációk
  - Ekvivalencia relációk tesztgeneráláshoz
- **Eszközök a tesztgeneráláshoz**

# További technikák: Szimbolikus végrehajtás

- A szimbolikus végrehajtás módszere:
  - Programutak felmérése
  - Bejárési feltételek összeállítása az útvonal mentén a forráskódban (byte kódban) talált feltételek alapján
  - Szimbolikus: változókra vonatkozó (logikai) kifejezések használata
- Tesztgenerálás az utakhoz:
  - Bemeneti értékek generálása a bejárési feltételeknek megfelelően:
  - Kényszerkielégítési probléma megoldás SMT megoldóval
- Kihívások:
  - Ciklusok: nagyszámú bejárható útvonal
  - Bonyolult aritmetika támogatása az SMT eszközben
  - Külső függőségek (pl. library-k) kezelése
- Eszközök
  - Java: Symbolic PathFinder (Java PathFinder alapokon)
  - Net: PEX
  - C: CUTE, EXE, KLEE

# További technikák: Tervkészítő (planner) használata

- Cél: Cél-orientált bejárás
- A tervkészítés elemei a tesztgeneráláshoz:
  - Kezdőállapot: Kiindulási (program)állapot
  - Célállapot: Elérendő (program)állapot
  - Operátorok (feltételek és hatások): Események/bemenetek hatására végrehajtott akciók, utasításblokkok
- Teszt: Terv a célállapot elérésére a kezdőállapotból
  - Operátor példányok halmaza
  - Részleges rendezési reláció az operátorok között
  - Ok-okozati kapcsolatok az operátorok között: feltételek és hatások
  - Operátorok változóinak behelyettesítése
- A terv teljes sorrendezéssel teszt szekvenciaként használható (linearizálás)

# További technikák: Evolúciós algoritmusok

- **Evolúciós algoritmus:**
  - Véletlen bejárás alapján „generált” teszt szekvenciák
  - Tesztkészlet módosítása: szekvencia mutáció, keresztezés (szekvencia részekkel)
  - Teszt kritériumoknak jobban megfelelő tesztkészlet megtartása és további módosítása
- **Teszt kritériumok:**
  - Vezérlési folyam alapú fedettségek (állapotok, útvonalak, forgatókönyvek integrációs teszteléshez)
  - Adatfolyam alapú kritériumok (all-defs, all-uses)
- **Eszközök:**
  - Java: DOTgEAr

# Példák automatikus tesztgeneráló eszközökre I.

- Tesztelés modell ellenőrzővel
  - FSHELL: C programokhoz
    - CBMC (korlátos modellellenőrző) generálja az ellenpéldát mint teszt szekvenciát strukturális tesztelési kritériumokhoz
  - BLAST:
    - Ellenpélda generálás adott teszt célhoz: Absztrakt teszt eset
    - Szimbolikus végrehajtás: Teszt adatok generálása
  - UPPAAL CoVer, TRON:
    - Valós idejű rendszerek modellezése: Időzített automaták
    - UPPAAL modell ellenőrző generálja a teszt eseteket
    - Konformancia reláció a teszteléshez:  
Relativised timed input-output conformance (RTIOCO)
      - Időkezelés nélkül konzisztens az IOCO relációval



# Példák automatikus tesztgeneráló eszközökre II.

- Üvegdoboz tesztelés specifikáció alapján
  - JET: JUnit váz generálása JML elő- és utófeltételek alapján
    - Előfeltétel: Véletlen teszteléshez kötöttséget ad
    - Utófeltétel: Test oracle generálható
  - DART, CUTE, jCUTE, EXE
    - Adott állapothoz vezető bemeneti szekvencia: A feltételeket tartalmazó **kényszerkielégítési probléma** megoldása és szimbolikus végrehajtás
    - Feltételek a SAT bemenetéhez hasonlóan generálhatók
  - SpecExplorer (C#):
    - Spec# specifikáció alapján modell automata képzése (dinamikusan)
    - Bejárás: **Gráfelméleti**, legrövidebb út, vagy véletlen bejárás
    - Konformancia reláció modell és program között: Alternating simulation
  - DOTgEAr (Java):
    - Adatfolyam alapú kritériumok szerinti tesztelés is (all-defs, all-uses)
    - **Evolúciós algoritmussal**, véletlen bejárás alapján indítva és módosítva

# Példák automatikus tesztgeneráló eszközökre III.

- Tesztelés absztrakt adattípusok alapján

- Absztrakt adattípus definícióban szereplő axiómák alapján generált tesztesetek

- Axió

Absztrakt adattípusok: hordozó halmaz és műveletek

- Spe

```
Type Boolean is
```

```
  sorts Bool
```

```
  opns
```

```
    false, true : -> Bool
```

```
    not : Bool -> Bool
```

```
    and : Bool, Bool -> Bool
```

```
  eqns
```

```
    forall x, y: Bool
```

```
  ofsort Bool
```

```
    not(true) = false;
```

```
    not(false) = true;
```

```
    x and true = x;
```

- S

- A

- A

- T

- C

- T

# Példák automatikus tesztgeneráló eszközökre III.

- Tesztelés absztrakt adattípusok alapján
  - Absztrakt adattípus definícióban szereplő axiómák alapján generált tesztesetek
  - Axiómákban szereplő változóknak értékadás
    - Ekvivalencia osztályok, szélső értékek
- Speciális modellezési nyelvek támogatása
  - STG: LOTOS specifikációs nyelv
  - AGATHA: UML, SDL, STATEMATE modellek
    - Kényszerkielégítési probléma megoldása (változók kezelése):  
útvonal bejárás feltételek generálása
  - Autolink: SDL és MSC specifikáció alapján
  - TDE/UML: Fedettségi kritériumok és kényszerek megadhatók
  - Conformiq: UML (állapottérkép) modellekhez
  - T-Vec, DesignVerifier, Reactis, AutoFocus: Simulink modellekhez

# Összefoglalás

- **Modell alapú tesztgenerálás**
  - Fedési kritériumok teljesítéséhez
    - Vezérlés-orientált: állapotok, átmenetek fedése
    - Adatfolyam-orientált: def-use fedéshez
  - Hibamodell alapján (mutáció kimutatásához)
    - k-ekvivalencia reláció szerint
    - IOCO reláció szerint
- **Eszközök**
  - Direkt (gráfelméleti) algoritmusok
  - Modellellenőrzők: Ellenpélda generálása
  - SAT megoldók: Helyettesítési értékek generálása
  - Szimbolikus végrehajtás: Kényszerkielégítés bejáráshoz
  - Planner algoritmusok: Cél-orientált bejárás generálása
  - Evolúciós algoritmussal történő tesztgenerálás