

A modellellenőrzés alkalmazásai: Szintézis és tesztgenerálás modellellenőrzővel

dr. Majzik István

dr. Micskei Zoltán

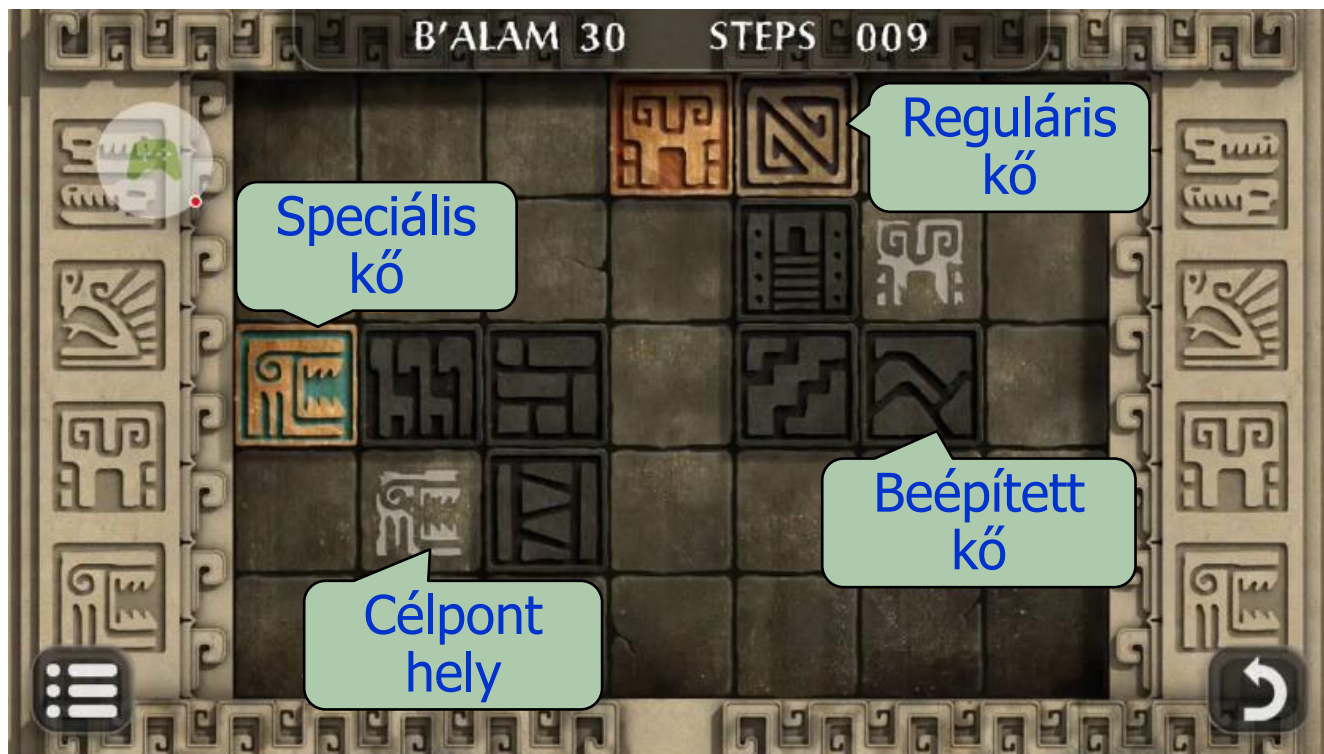
BME Méréstechnika és Információs Rendszerek Tanszék

Szintézis feladatok megoldása modellellenőrzővel

Bevezetés: Szintézis feladat

- Cél: Egy lépéssorozat meghatározása a probléma modelljében
 - Gyártási folyamat: milyen lépésekkel készül el a termék?
 - Protokoll, számítás: milyen lépésekkel kapható eredmény?
 - Játék, rejtvény: milyen lépésekkel van megoldás?
- Modellellenőrző: minden lehetséges lépéssorozatot vizsgál; ezek közül kell a számunkra érdekeset „lekérdezni”
 - Ha ellenpélda keresés a modellellenőrző célja:
A keresett lépéssorozat legyen megfeleltethető egy várt ellenpéldának
 - Ha tanú (witness) keresés a modellellenőrző célja:
A keresett lépéssorozat elérhetősége legyen vizsgálva
- Egyszerű esetek: Lépéssorozat célállapotának megadásával
 - Lépéssorozat generálás ellenpélda kereséssel: $\neg EF <\text{célállapot}>$
 - Lépéssorozat generálás tanú kereséssel: $EF <\text{célállapot}>$
- Legrövidebb, leggyorsabb lépéssorozat meghatározása
 - Az ellenpélda vagy tanú elvárt tulajdonságának beállítása
 - Pl. UPPAAL: legrövidebb vagy leggyorsabb trace meghatározása

Mintapélda: A Cryptica játék megoldása



- Cél: Speciális kövek a célpontjaikba kerüljenek
- Egy lépés: Minden speciális és reguláris követ (amit lehet) **balra** vagy **jobbra** vagy **felfelé** vagy **lefelé** kell mozgatni
 - A pálya széle vagy beépített kő egy-egy követ megállít, de a többi kő mozgatását nem hiúsítja meg

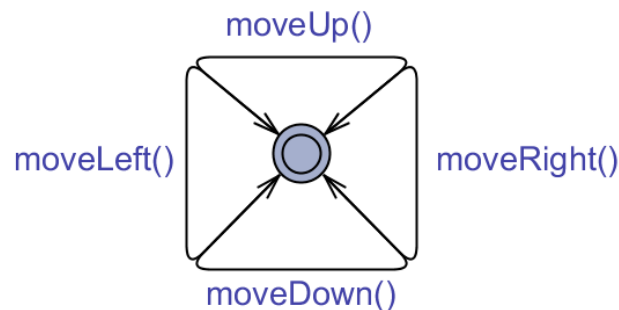
Mintapélda: A Cryptica játék modellje

- Adatstruktúrák

- **Pálya** modellje: Beépített reguláris kövek és célpontok
- **Játékállás** modellje: Speciális és reguláris kövek változó elhelyezkedése

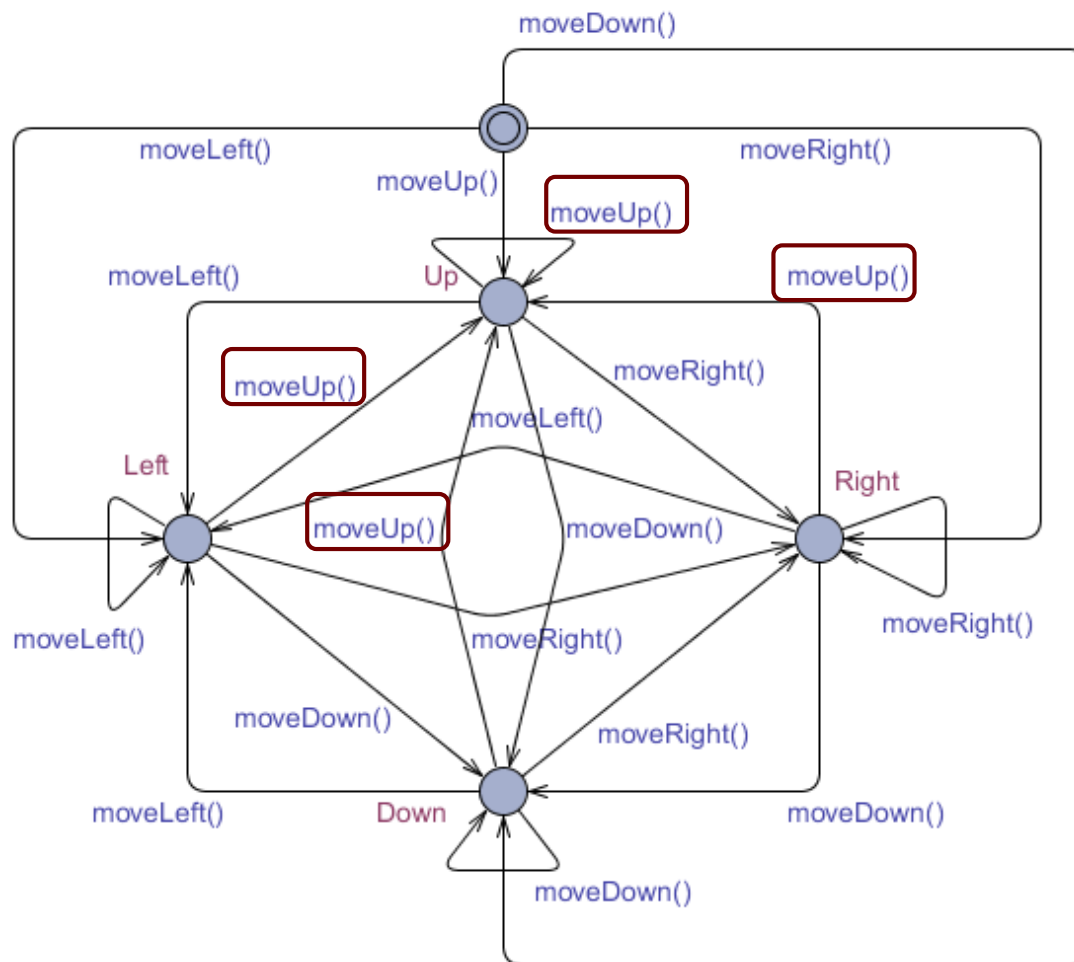
- Dinamikus működés: Lehetséges lépések felvétele

- Lépés balra / jobbra / felfelé / lefelé:
játékállást módosító **függvények** a pálya modellje alapján
- A modellben megjelenik **minden lehetséges lépés**;
a kívánt lépéssorozatot a modellellenőrző „keresi” majd
- Legegyszerűbb automata:



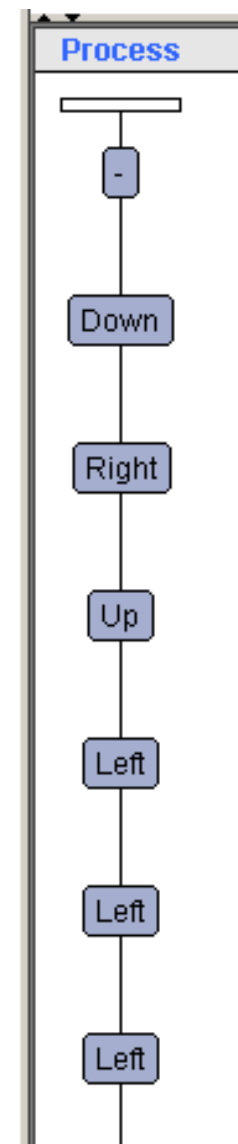
Mintapélda: A Cryptica játék modellje

- Célszerűbb modell: Lépések követhetővé tétele
 - A legutóbbi lépést jelzi a lépéssel elért állapot neve



Mintapélda: Cryptica játék modellellenőrzése

- Modellellenőrzés: Kirakható-e a játék?
 - Célállapot meghatározás: `isCorrectDistribution()` logikai függvény (csak akkor igaz, ha a speciális kövek a helyükön vannak)
 - Temporális logikai követelmény tanú trace-hez: $E\langle \rangle \text{isCorrectDistribution}()$
- Legkevesebb hány lépésben kerülhetnek helyükre a speciális kövek?
 - A legrövidebb diagnosztikai trace (tanú trace) keresése a kirakáshoz
- A kirakás lépései a szimulátorban megjelenített állapotszekvencián követhetők

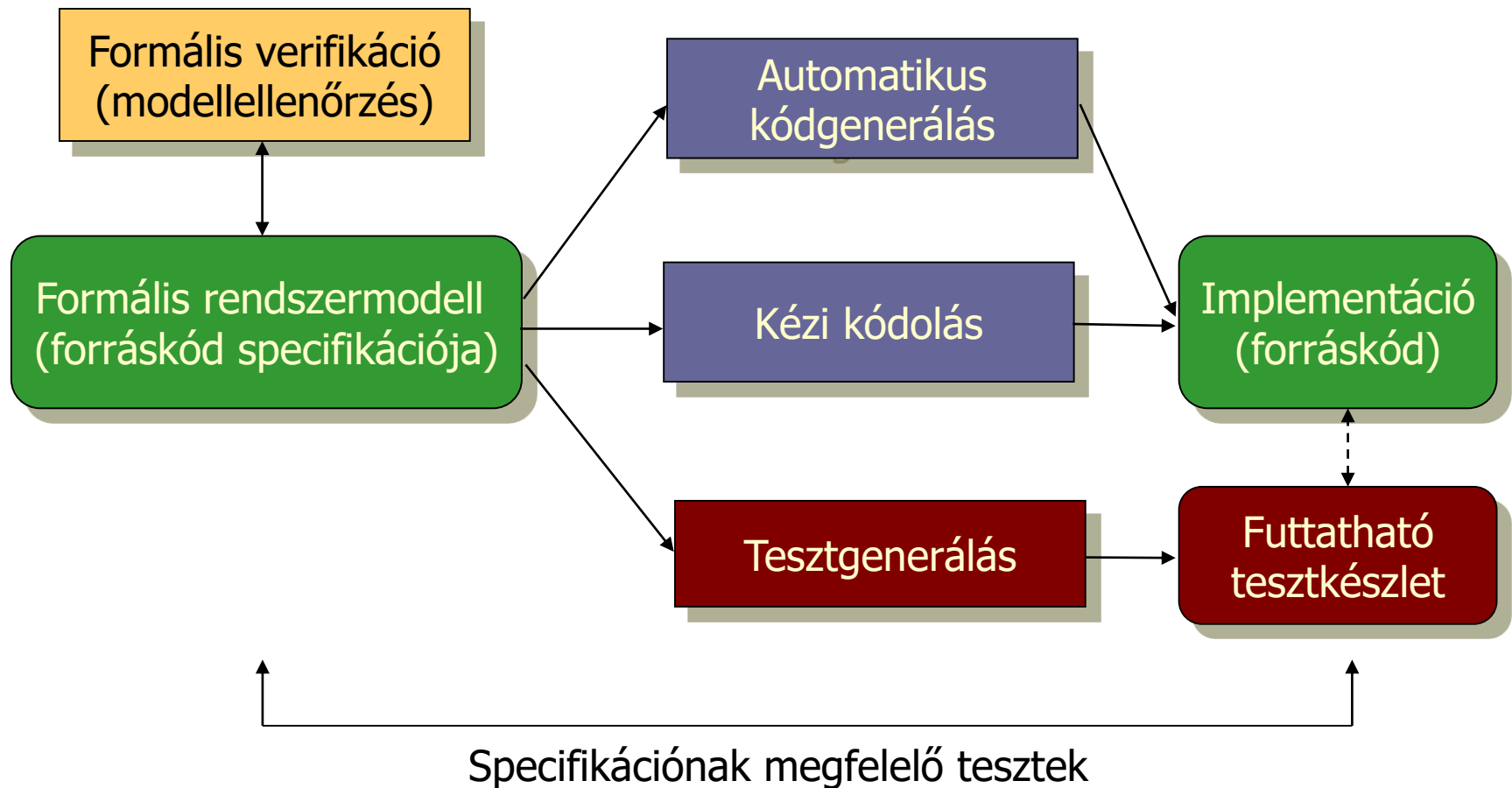


Elvárt lépéssorozat specifikálása

- Célállapot megadása: Cél vezérlési hely vagy változóértékek
 - Temporális kifejezés: $E \langle \rangle$ célállapot
- Közbenső lépések megkötése: Adott vezérlési hely mindenképpen érintve legyen
 - Temporális kifejezés: $E \langle \rangle (\text{közbenső} \wedge (E \langle \rangle \text{célállapot}))$
- Probléma: Ha korlátozott a CTL, pl. UPPAAL esetén: Csak a formula elején lehet temporális operátor
 - Sok problémát megold segédváltozók bevezetése
 - Példa: **belépett** logikai segédváltozó, ami csak a közbenső vezérlési helyre való belépéskor kap igaz értéket
 - Temporális kifejezés: $E \langle \rangle (\text{belépett} \wedge \text{célállapot})$
 - Ilyen módon megadhatók a lépéssorozathoz:
 - Tiltott vezérlési helyek (hová ne lépjen be)
 - Korlátozott számú belépés adott vezérlési helyre (számlálóval)
 - ...

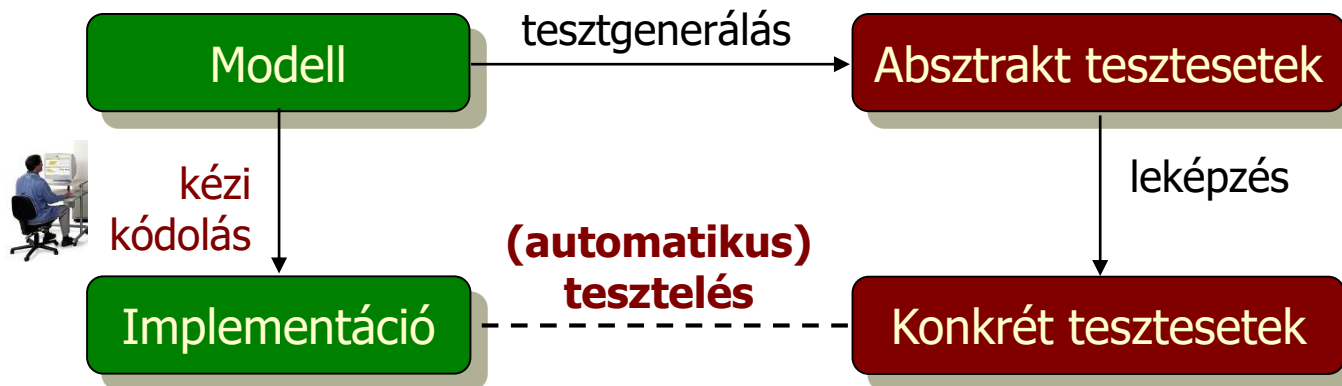
Tesztgenerálás modellellenőrzővel

Tesztgenerálás szerepe a modell alapú fejlesztésben



Generált tesztek használati esetei

- Kézi kódolás esetén: Konformancia ellenőrzés



- Automatikus kódgenerálás esetén: Validáció



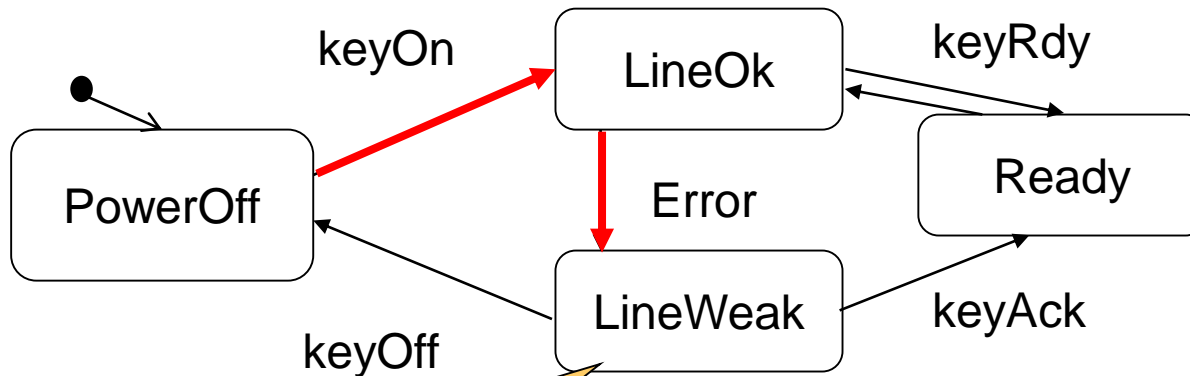
Előfeltételek és használat

- Állapot alapú, eseményvezérelt működés
 - KS, LTS, KTS, TA az alapszintű formalizmusok
- Fedési kritériumok szerinti tesztelés
 - Állapotfedés: A tesztekkel járunk be minden állapotot
 - Átmenetfedés: A tesztekkel járunk be minden átmenetet
- Alapötlet:
 - Egy teszt: Egy megfelelő lépéssorozat az állapottérben egy fedési célhoz (pl. adott állapot fedése)
 - Cél: A modellellenőrző keresse ezt az állapottérben
 - Irányítsuk úgy, hogy a modellellenőrző által generált ellenpélda legyen a teszteset ← lépéssorozat
 - Teszt bemenetek és elvárt kimenetek: A modell alapján

Hogyan használható a modellellenőrző?

- Állapot fedettség:
LineWeak állapotra

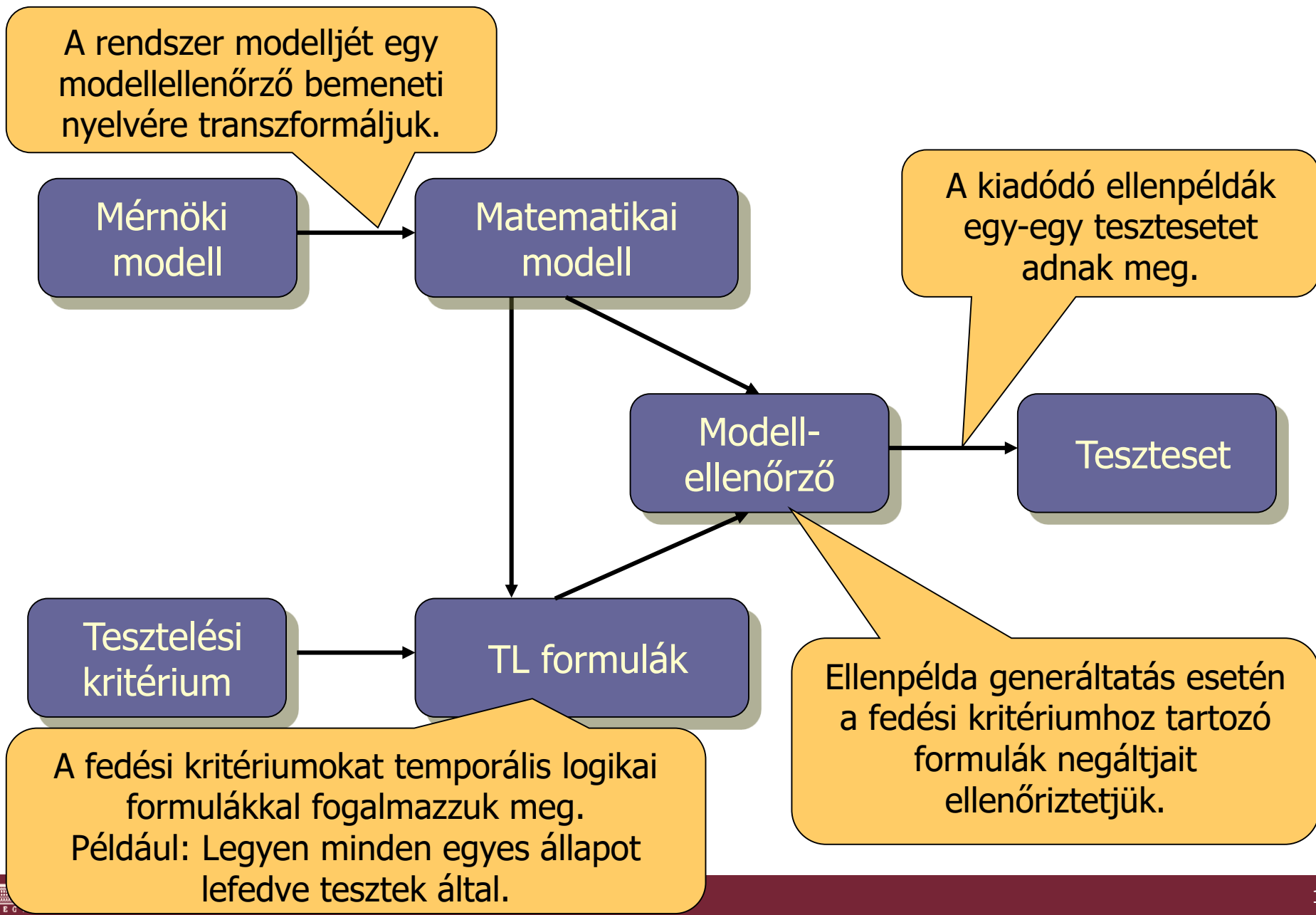
A modellellenőrző
ellenpéldával demonstrálja,
hogy a kritérium nem
teljesül, az állapot elérhető.



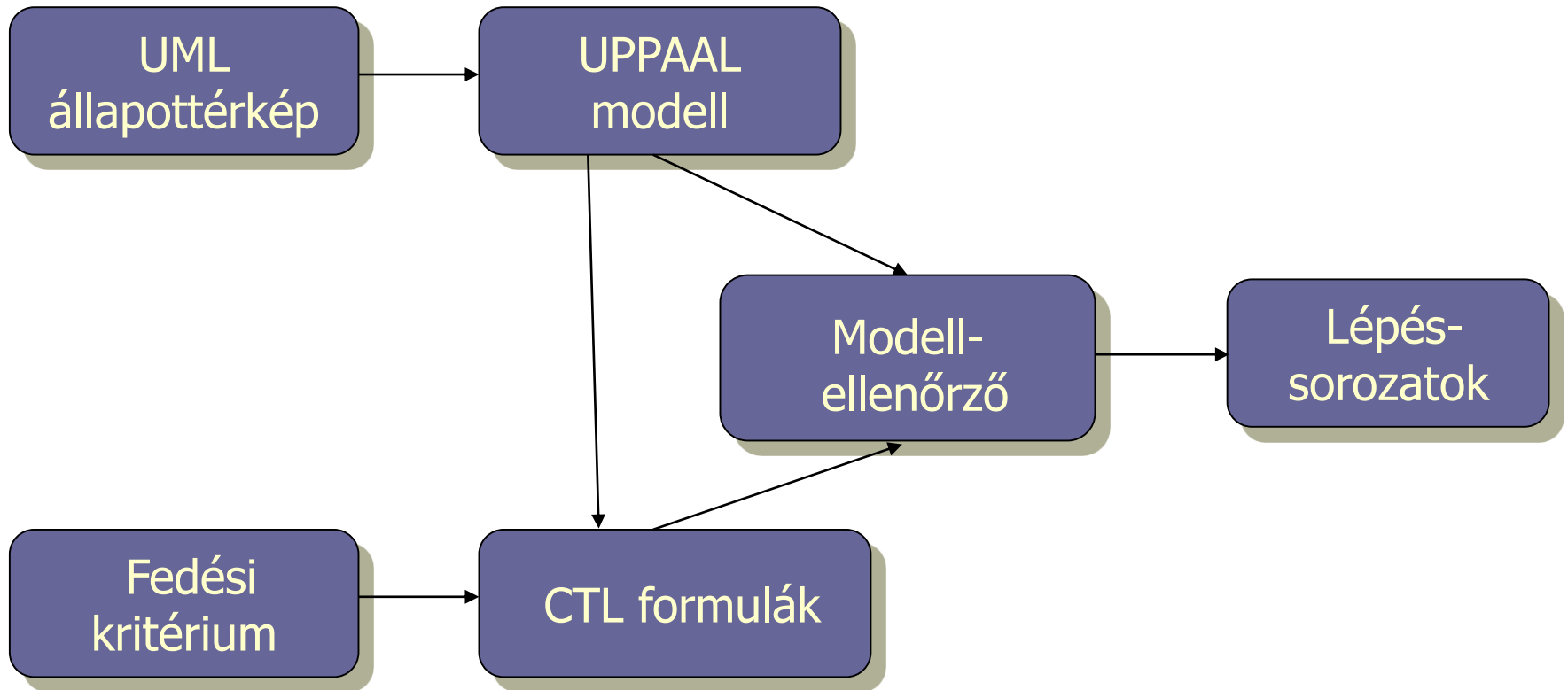
Kritérium megadása:
A LineWeak állapotot
nem lehet elérni:
→ EF LineWeak

Ez pontosan egy,
a LineWeak állapotot
lefedő tesztesetet ad.

Automatikus tesztgenerálás



Egy megvalósítás



Vezérlés alapú fedettségi kritériumok

Minden állapot illetve állapotátmenet azonosítva:
Egyedi $L(s)$ címke illetve (a) akció

- Állapot fedés: KS vagy KTS modellen

Minden s állapotra: $\neg EF L(s)$ vagy

$\neg EF (L(s) \wedge EF \text{ start})$

start egy megfelelő állapot címkéje
a következő teszt indításához

- Állapotátmenet fedés: LTS vagy KTS modellen

Minden t átmenetre, ahol $(s,a,s') \in \rightarrow$: $\neg EF (a)$

Korlátozások

- Modellellenőrző képességei
 - Csak egy ellenpéldát generál, ami nem feltétlenül a legrövidebb
 - Sok modellellenőrző konfigurálható:
 - Legrövidebb ellenpélda kérhető
 - Szélességi bejárás kérhető (így rövid teszteset lesz)
 - Mélységi bejáráshoz mélységkorlát megadható (teszt hossz korlát)
 - Rövidebb ellenpélda iteratíván kereshető (hosszadalmas lehet)
 - Legrövidebb illetve minimális tesztkészlet kiválasztása: NP-teljes
- Absztrakt és konkrét tesztesetek közt leképezés kell
 - Absztrakt teszteset: Lépéssorozat (ellenpélda) a modell alapján
 - Konkrét teszteset: Hívási szekvencia adott teszt környezetben
- Nemdeterminisztikus modellek esetén nehézségek
 - Generált teszt: egy lehetséges végrehajtás – lefut-e?

Példa: Tesztgenerálási eredmények (állapotfedéshez)

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

- Mobiltelefon vezérlését leíró modell
- 10 állapot, 21 átmenet

Paraméterek a SPIN modellellenőrzőhöz:

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

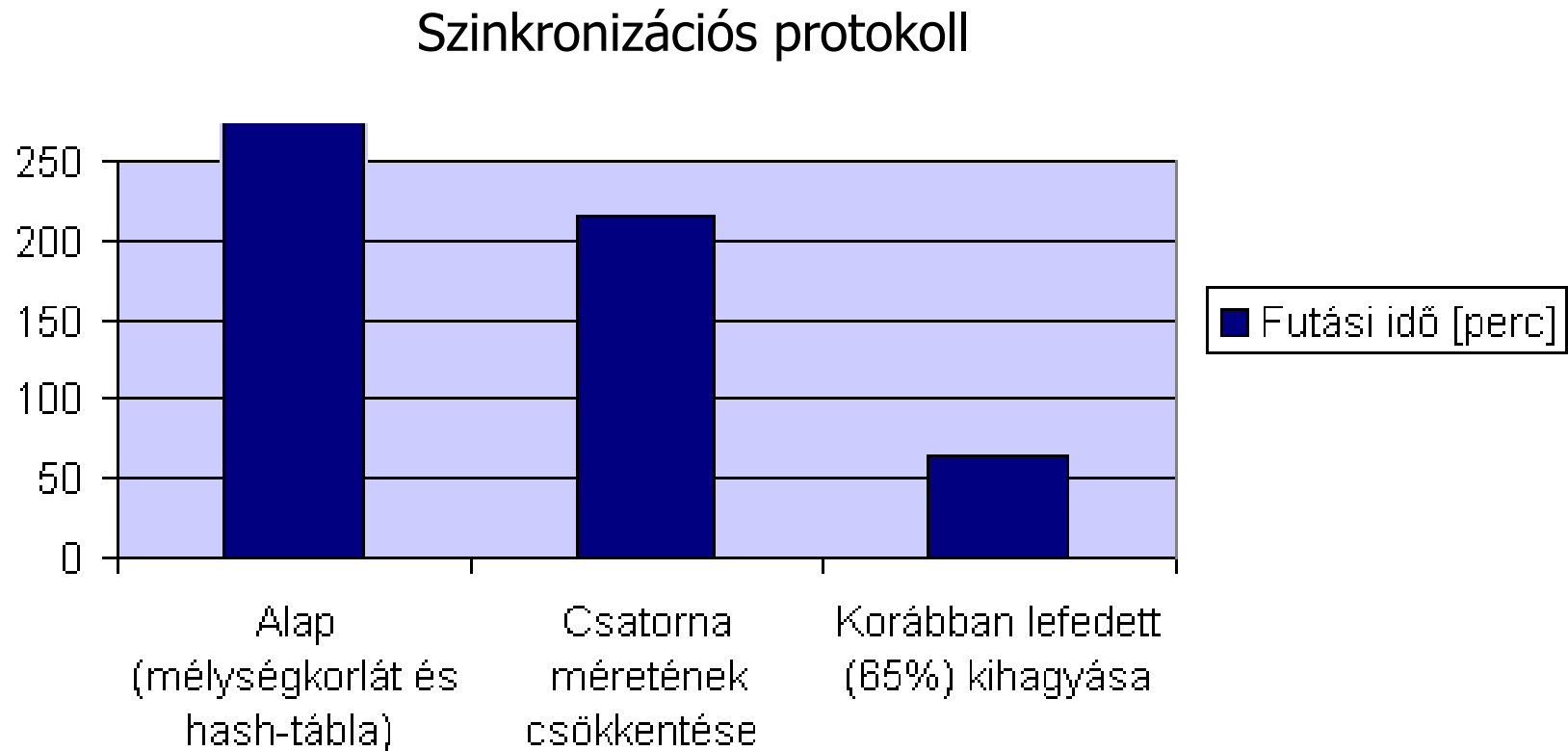
Jó ötletek:

- Mélységi kereséshez korlát megadása
- Rövidebb ellenpéldák közelítő iteratív keresése

Példa: Szinkronizációs protokoll tesztelése

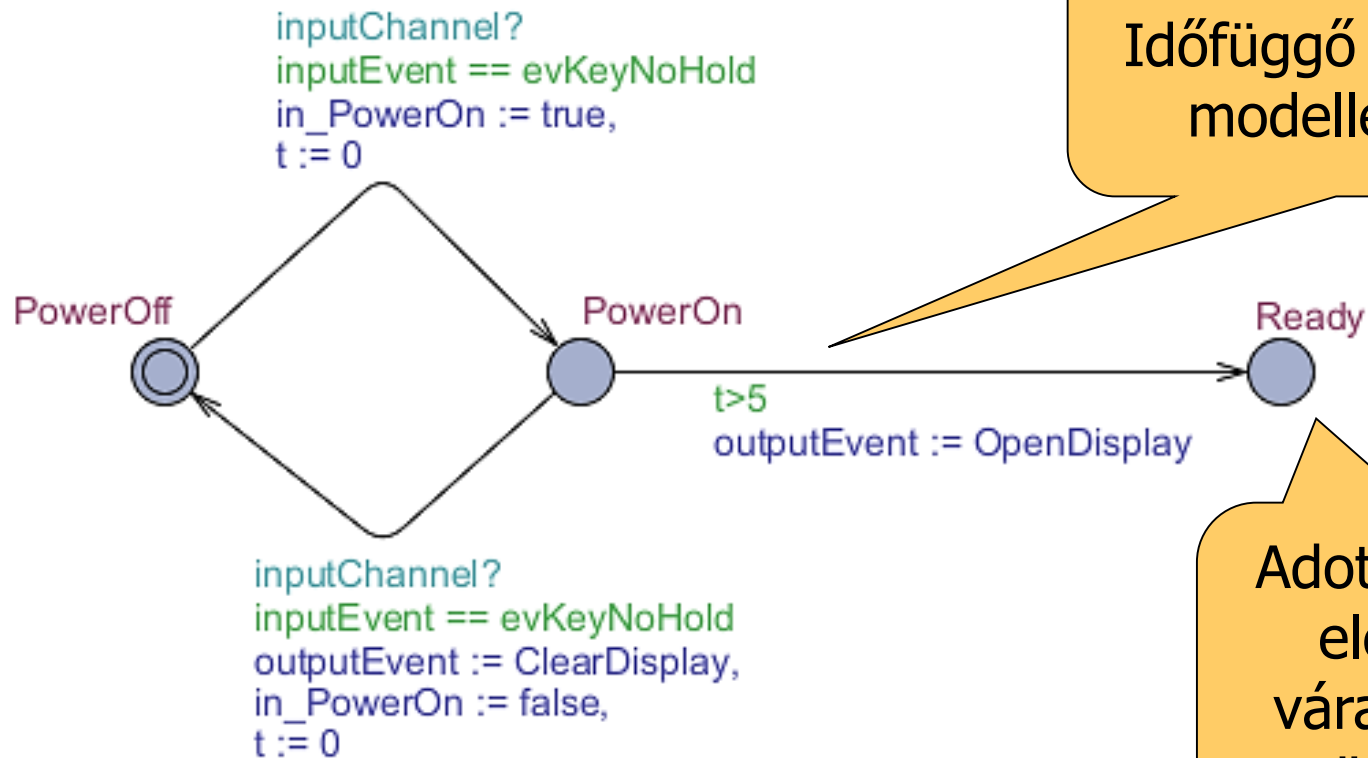
- Bitek szinkronizálása egy elosztott rendszerben
 - 5 objektum, 31 állapot, 174 átmenet
 - $2e+08$ bejárandó állapot
- Nagy állapottér kezelése
 - Mélységkorlát bevezetése a kereséshez
 - Szűkítések a modellben:
 - FIFO kommunikációs csatorna méretének korlátozása
 - Már lefedett kritériumok figyelése és kihagyása
- További ötletek
 - Mélyen fekvő állapotok lefedése előbb
 - Állapottér közelítő reprezentálása (hash-tábla)
 - Adott állapotokat nem jár be a keresés során

Példa: Teljes állapotfedésű tesztek



Tesztgenerálás időfüggő viselkedés ellenőrzéséhez

- Időzített automaták használata
- Modellellenőrző: UPPAAL



Óraváltozók:
Időfüggő viselkedést
modellezhetünk

Adott fedési cél
eléréséhez
várakozásra is
szükség lehet

Példa: Az UPPAAL verifyta által generált trace

State:

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

```
t=0 inputEvent=28 outputEvent=14 in_PowerOn=0 #depth=5
```

Delay: 6

A teszthez szükséges
időzítési viszonyok is
szerepelnek a generált
lépéssorozatban

State:

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

```
t=6 inputEvent=28 outputEvent=14 in_PowerOn=1 #depth=5
```

Transitions:

```
input.sending->input.sendInput { 1, inputChannel!, 1 }  
mobile2.CallWait->mobile2.VoiceMail { inputEvent ==  
  evKeyYes && t > 5 && in_PowerOn, inputChannel?, 1 }
```

Tesztgenerálás korlátos modellellenőrzővel

- Logikai függvény konstruálása:
 - Állapottér kibontása k lépésben a kezdőállapotból
 - Teszt kritérium megadása: TG formula, pl.:
 - Adott állapot, állapotátmenet elérése
 - Adott modellrészlet bejárása, ...
- SAT megoldónak átadott formula:

FSHELL FQL nyelv teszt célokhoz:
in /code.c/
cover @line(6),@call(f1)
passing @file(c1.c) \ @call(f2)

$$\text{SAT} \left(I(s^0) \wedge \text{path}(s^0, s^1, \dots, s^k) \wedge TG \right)$$

Kezdő-
állapot

Állapottér
kibontása

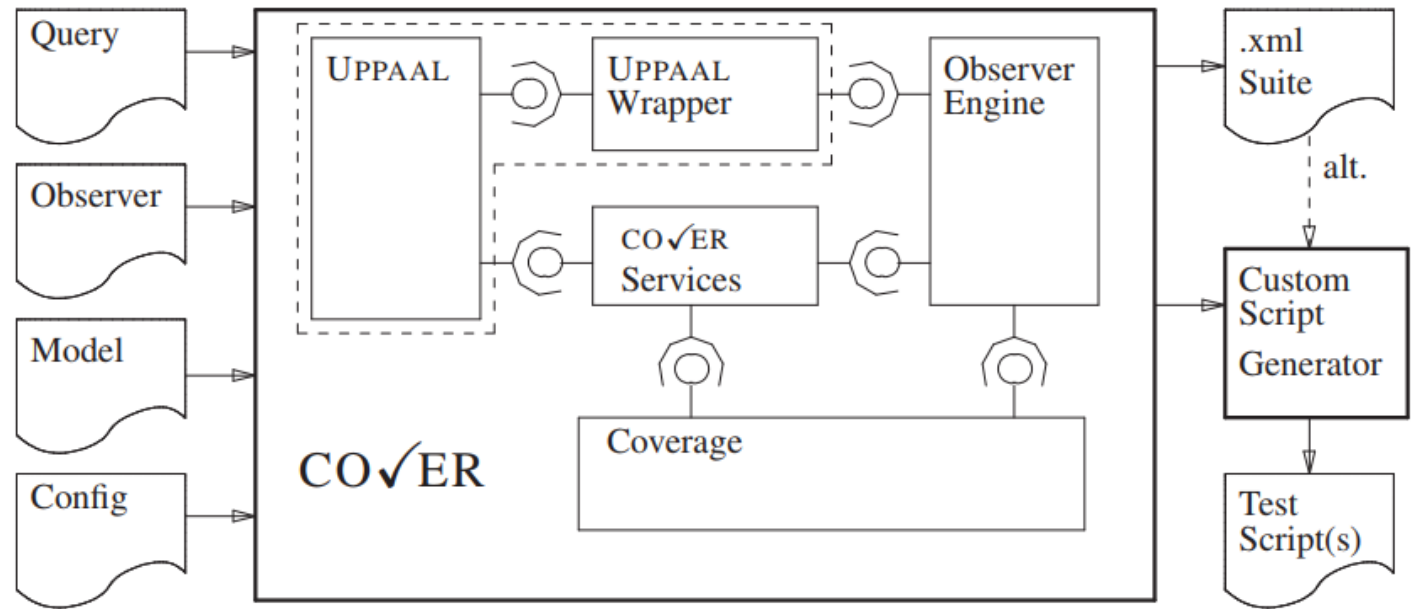
Teszt
cél

- Ha található behelyettesítés, akkor az egy tesztet ad:
 - A teszt teljesíti a TG kritériumot
 - A legrövidebb teszt 0-ról induló iteráció során található meg

Példa eszköz: UPPAAL CoVer

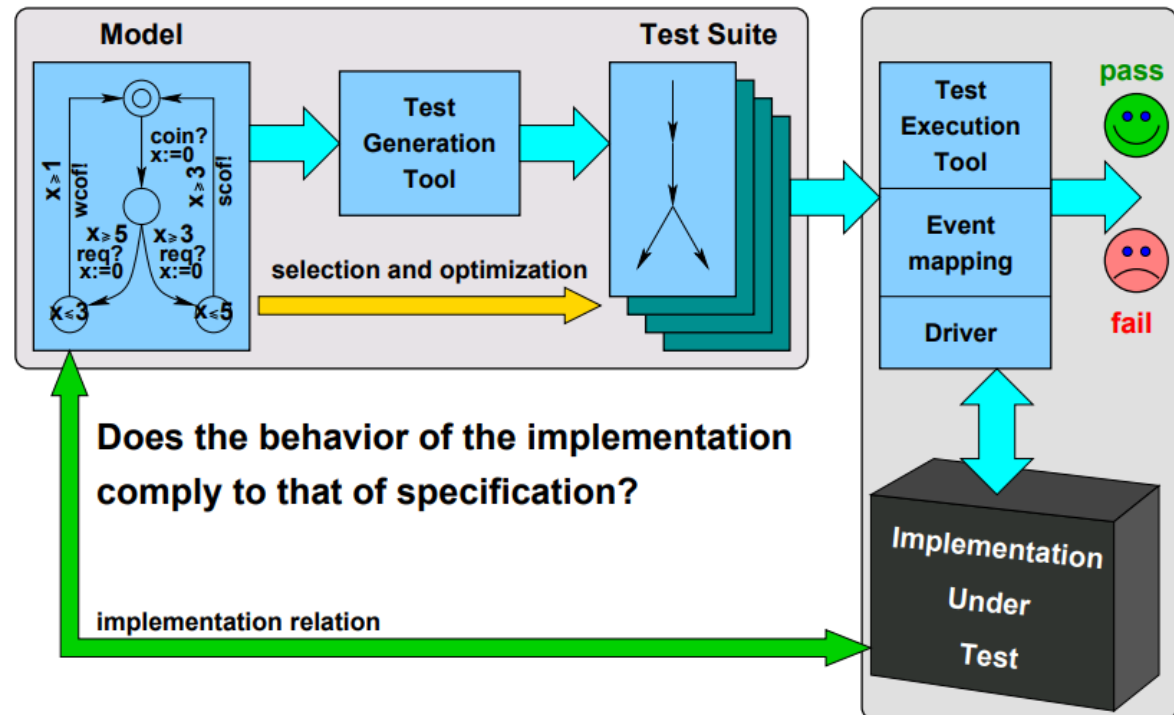
- UPPAAL COVER

- „UPPAAL CoVer is a tool for creating test suites from UPPAAL models with coverage specified by coverage observers a.k.a. observer automata.”



Példa eszköz: UPPAAL TRON

- UPPAAL TRON: Testing Real-time Systems Online
 - „UPPAAL TRON is a testing tool, based on UPPAAL engine, suited for black-box conformance testing of timed systems, mainly targeted for embedded software commonly found in various controllers.
 - By online we mean that tests are derived, executed and checked simultaneously while maintaining the connection to the system in real-time.”



Összefoglalás

- Tesztgenerálás fedési kritériumokhoz
 - Állapotok fedése
 - Átmenetek fedése
 - ...
- Klasszikus modellellenőrzők használata
 - Fedési kritériumhoz temporális logikai kifejezőkészlet
 - Ellenpéldák adják a teszteseteket
 - A modellellenőrző konfigurálása fontos
- SAT alapú (korlátos) modellellenőrző használata
 - Fedési kritérium (teszt cél) mint predikátum
 - SAT eredménye (behelyettesítés) adja a tesztesetet