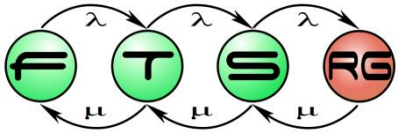
Modellezés, modellellenőrzés

Darvas Dániel, Horányi Gergő

Modellalapú tervezés és kódgenerálás szakkör



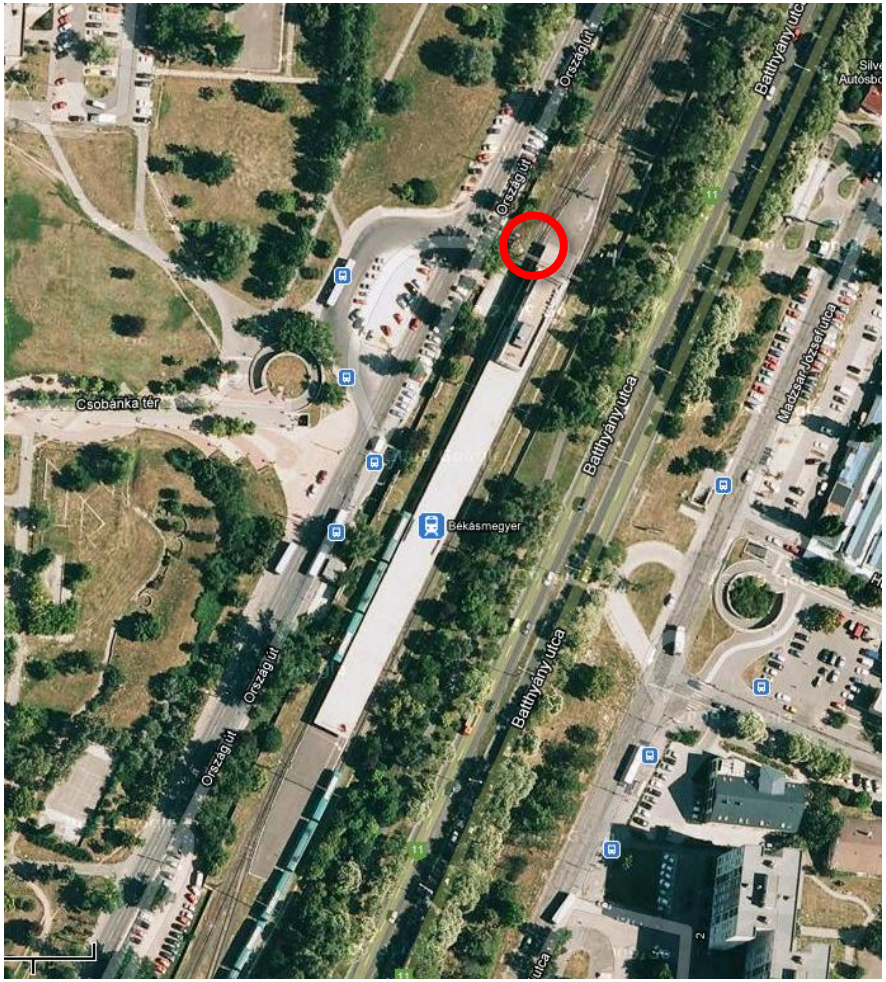
Modellezés

- Mi az a modell?

- Mi nem a modell?

Ami biztosan nem modell: **a valóság**

Békásmegyer, HÉV-állomás



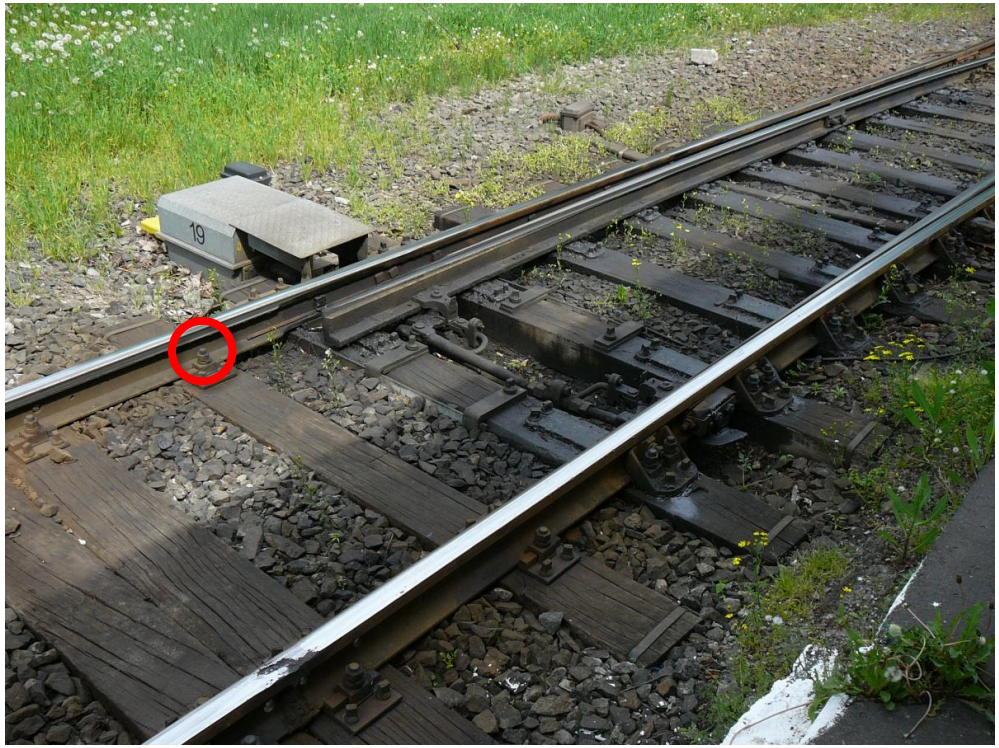
HÉV-állomás

Adatok:

- peron hossza: 268,4 m
- peron szélesség: 12,3 m
- tszf. magasság: 104 m
- építés ideje: 1975

Ami biztosan nem modell: **a valóság**

Békásmegyer, HÉV-állomás



19-es váltó

Adatok:

- sín típusa: MÁV 48,5 kg/fm
- állítófeszültség: 400 V
- max. állítóáram: 130 A
- vezérlő bizt. ber.: Thales Elektra2
- kampózár: van
- alj: fa

Ami biztosan nem modell: **a valóság**

Békásmegyer, HÉV-állomás

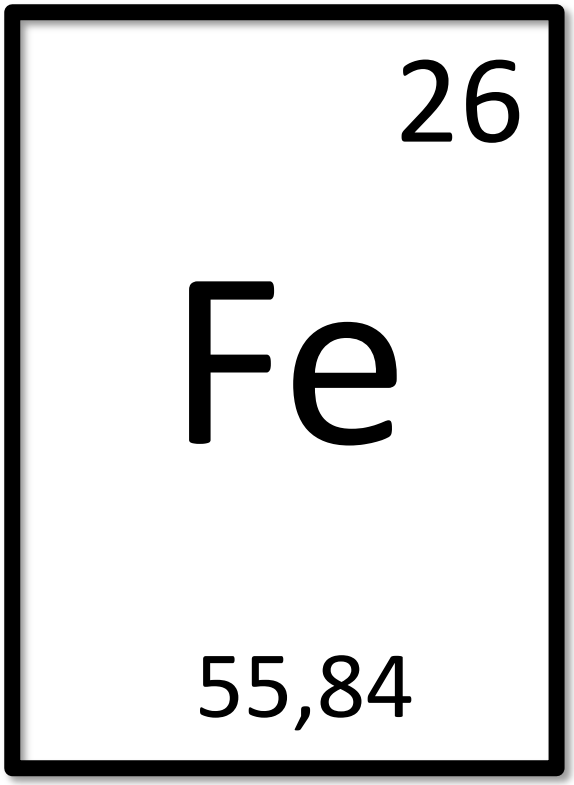
Sínleerősítés

Adatok:

- rendszer: GEO
- alátétlemez: kétbordás
- Grower-gyűrű: van
- síncsavar típusa: KL MÁVSZ 2936:1996
- szorítócsavar mérete: M22
- sínvándorlásgátlás: van
- anyaga: vas

Ami biztosan nem modell: **a valóság**

Békásmegyer, HÉV-állomás



Vas

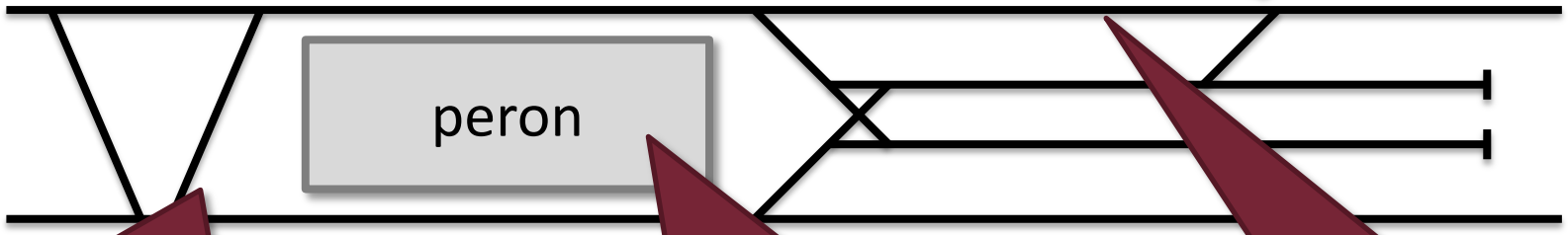
Adatok:

- vegyjel: Fe
- atomtömeg: 55,84 g/mol
- elektronszerkezet: [Ar] $3d^6$ $4s^2$
- olvadáspont: 1538 °C
- atomsugár: 140 pm
- fajlagos ellenállás: 96 nΩm
- CAS-szám: 7439-89-6

- De mi az, ami tényleg érdekelt minket?
  - pl. hány peron melletti vágány van?

Kettő.

- az állomás **modellje**:



Melyik a 19-es váltó?

Be tudna itt fordulni egy vonat?

Milyen a peron burkolata?

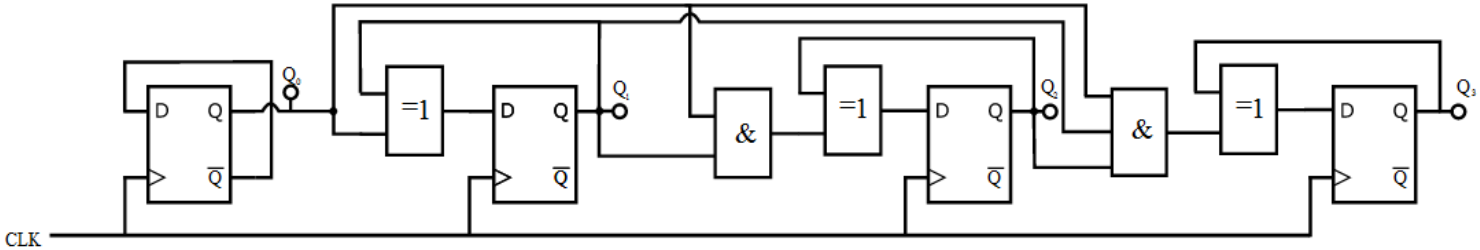
Mekkora a nyomtávolság?

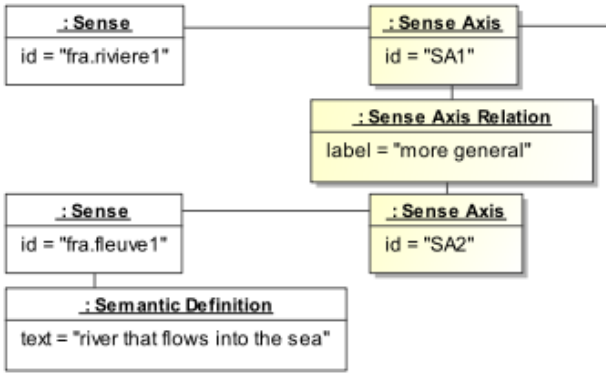
Modell

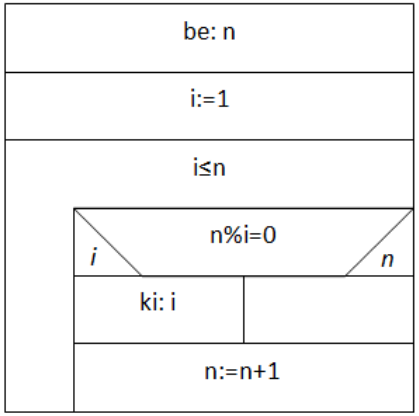
- Mi a modell?
  - „A valóság egy részletének egyszerűsített képe” (Rendszermodellezés tárgy)
  - (de: modell ≠ diagram!)
- Miért modellezünk?
  - kezelhető (véges) komplexitás elérése
  - a minket érdeklő aspektusra koncentrálhatunk
    - → áttekinthető lesz!

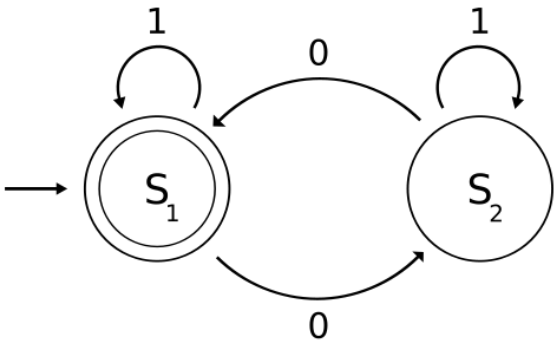
Modellezés

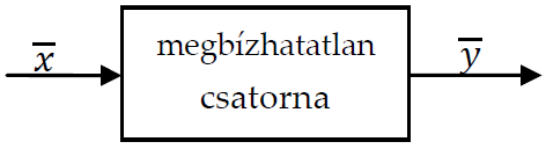
- Milyen modellekkel találkoztunk már?

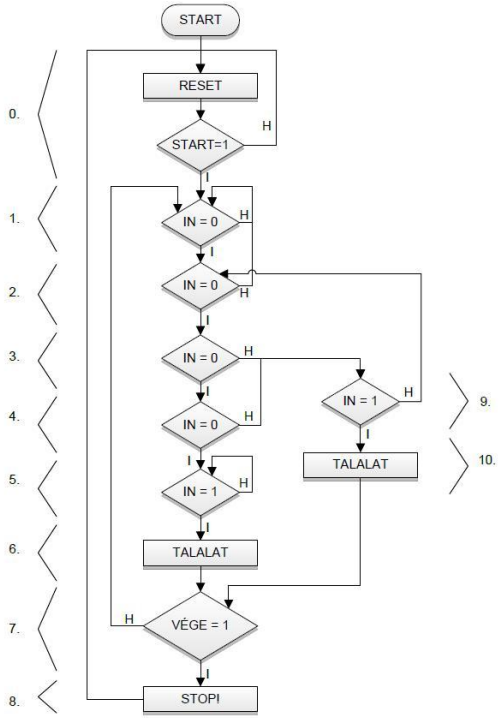






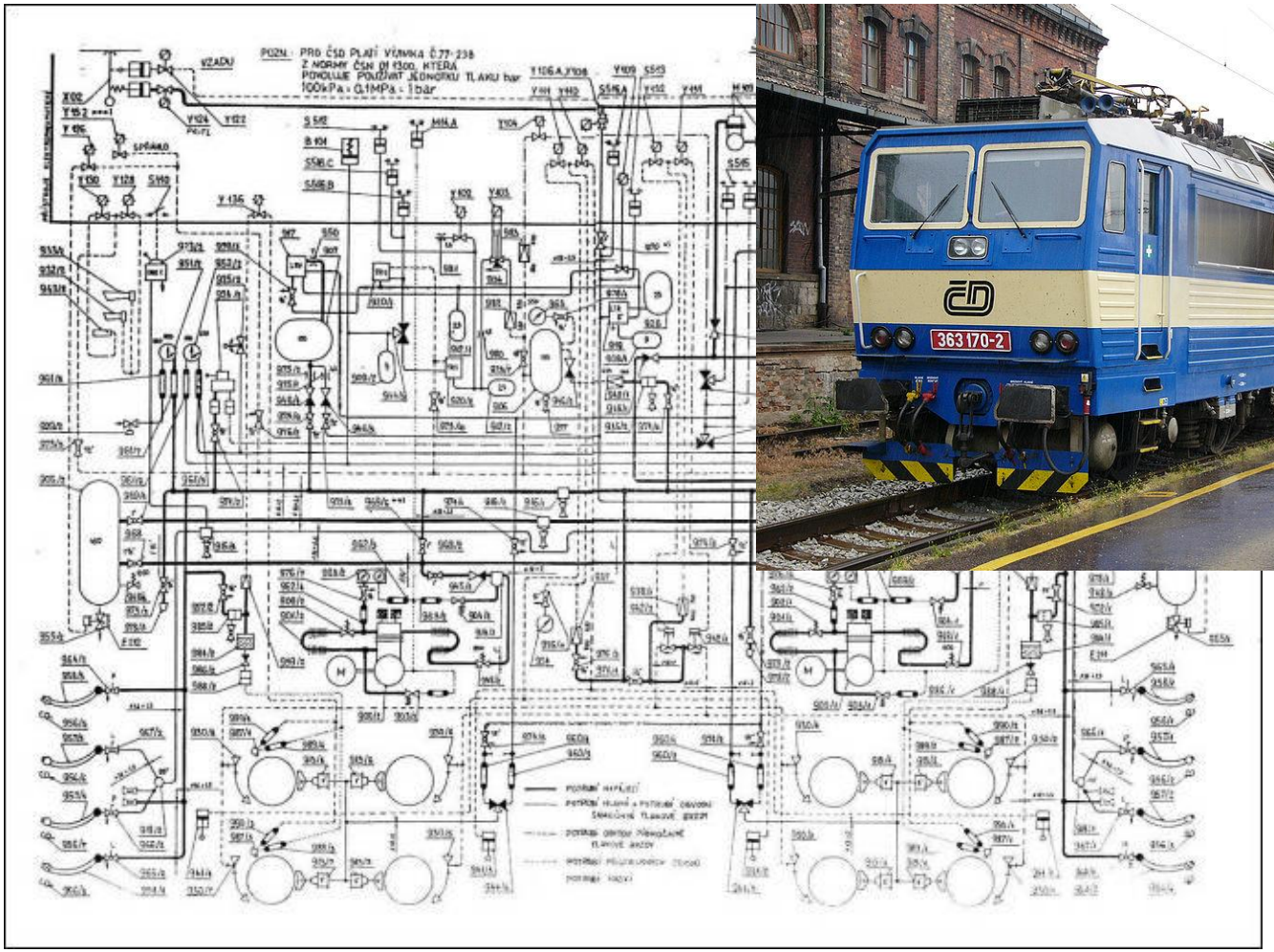






A valóság *egyszerűsített* képe?

- ČD 363 sorozatú mozdony fékrendszere



Forrás: www.kbsz.hu

Modellek felhasználása

- dokumentáció
  - A teljes valóság nem írható le egy dokumentációba, de nem is szükséges.
- **analízis**
  - pl. hibák keresése
- **szimuláció, kísérlet**
- származtatás
  - pl. kódgenerálás → később

Részben a Rendszermodellezés tárgy diái alapján: www.inf.mit.bme.hu/edu/courses/remo

Verifikáció

- ma főleg erre koncentrálunk
  - használt formalizmus: „UPPAAL-modell”
- miért ellenőrizzük a modelljeinket?
  - minden fejlesztő hibázik
  - bizonyos hibák különösen nehezen kiszűrhetőek (intuitívan nem látszanak)
  - becsült hibaszám jó fejlesztés és tesztelés mellett: ≈ 10 hiba / KLOC (Majzik I.: Szolgáltatásbiztonságra tervezés)
  - pl. katasztrofális vasúti események elfogadható gyakorisága: ≈ 1 / 10^9 működési óra (CENELEC 50126, SIL4)

Ariane-5 Flight 501

Ariane-5 Flight 501

- **Ariane-5** első tesztrepülése (1996. jún. 4.)
  - http://www.youtube.com/watch?v=c9Hf4qTxdxs
  - 40 s után felrobbant (önmegsemmisítés)
  - kár: legalább US$ 370,000,000 (más adat: 8 Mrd $)
  - ok: szoftverhiba (mindkét fedélzeti számítógépen – az ellen nem véd…, teszten nem jött ki)

  - felhasználták az Ariane-4 szoftverét
  - ebben volt egy float-int konverzió
  - az Ariane-5 gyorsulása nagyobb volt az Ariane-4-énél
  - a 64 bites float gyorsulási adat nem fért el a 16 bites előjeles int változóban, ez kezeletlen kivételt váltott ki

Ariane-5 Ada kód

```
end if;
L_M_DON_32 := TDB.T_ENTIER_32S ((1.0/C_M_LSB_DON) *
                                G_M_INFO_DERIVE(T_ALG.E_DON));
if L_M_DON_32 > 32767 then
   P_M_DERIVE(T_ALG.E_DON) := 16#7FFF#;
elsif L_M_DON_32 < -32768 then
   P_M_DERIVE(T_ALG.E_DON) := 16#8000#;
else
   P_M_DERIVE(T_ALG.E_DON) := UC_16S_EN_16NS(
      TDB.T_ENTIER_16S(L_M_DON_32));
end if;

P_M_DERIVE(T_ALG.E_DOE) := UC_16S_EN_16NS (TDB.T_ENTIER_16S
                                ((1.0/C_M_LSB_DOE) *
                                G_M_INFO_DERIVE(T_ALG.E_DOE)

L_M_BV_32 := TDB.T_ENTIER_32S ((1.0/C_M_LSB_BV) *
                                G_M_INFO_DERIVE(T_ALG.E_BV));
if L_M_BV_32 > 32767 then
   P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then
   P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
   P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M
end if;

P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS (TDB.T_ENTIER_16S
                                ((1.0/C_M_LSB_BH) *
                                G_M_INFO_DERIVE(T_ALG.E_BH)))

end LIRE_DERIVE;
--$finprocedure

--(
procedure LIRE_SEUIL (P_M_SEUIL : out TDB.T_ENTIER_16NS) is
--)
```

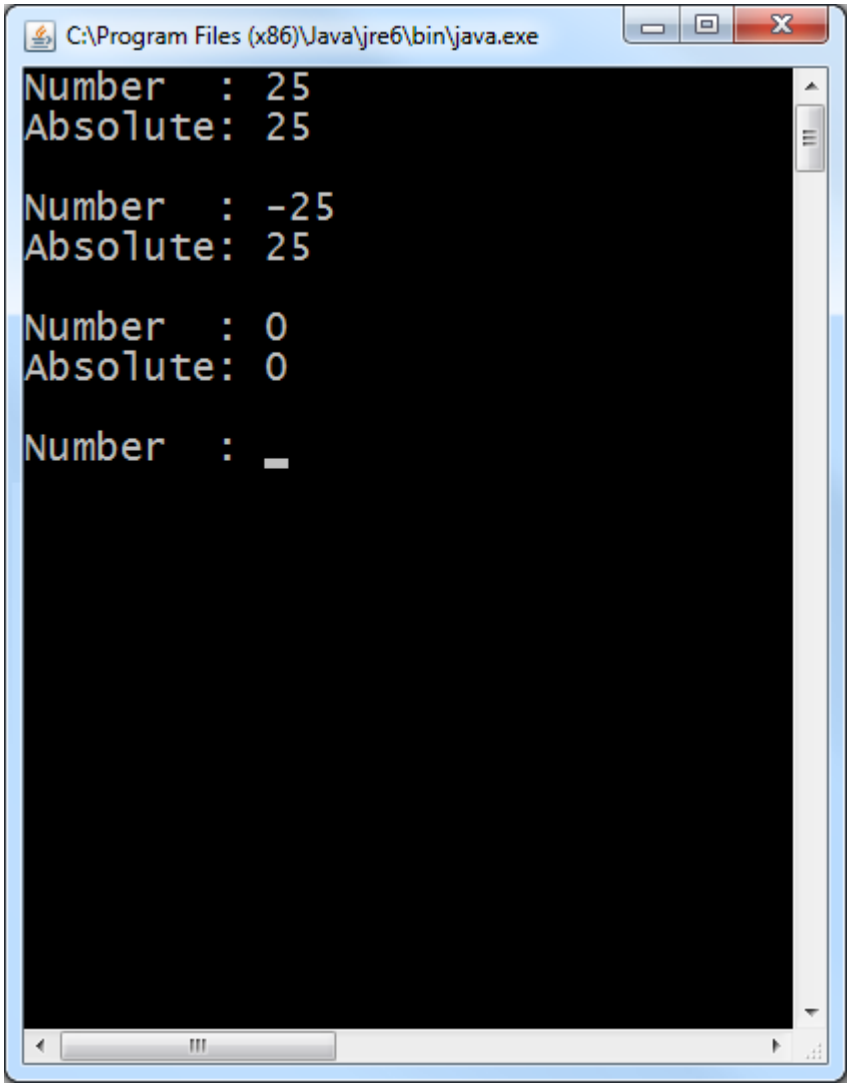
≈70 KLOC

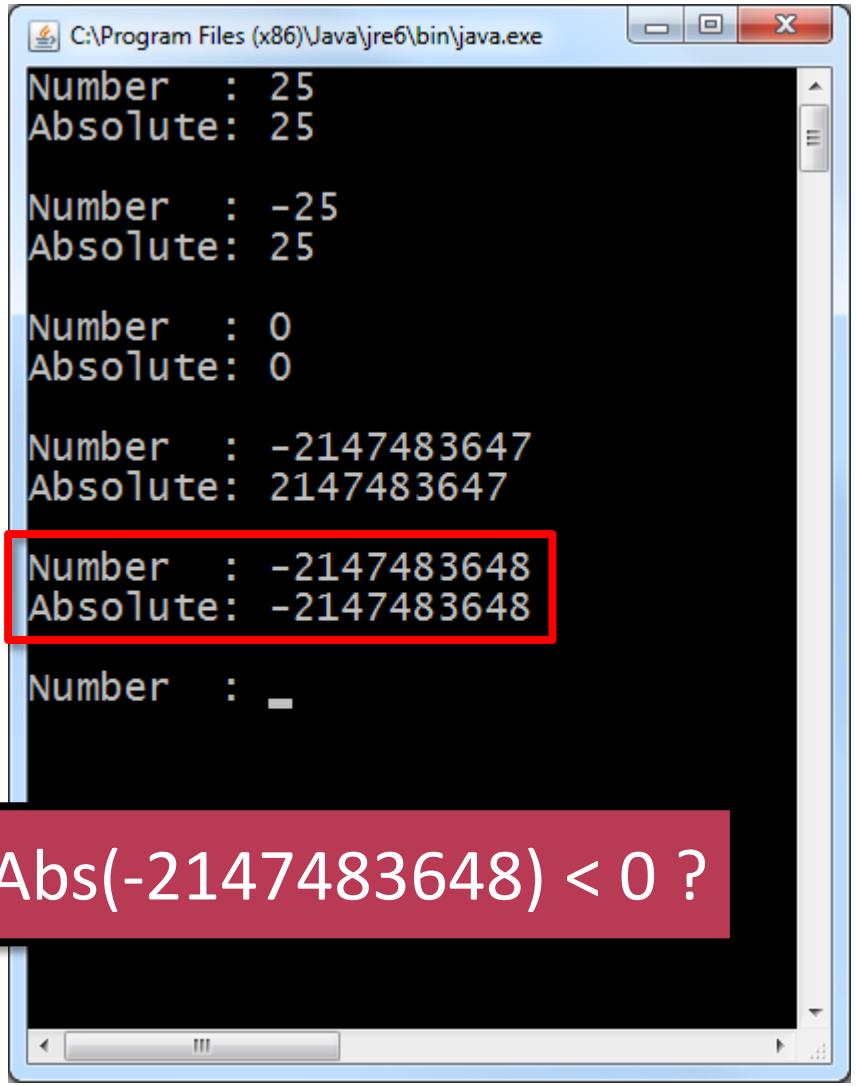
Nemtriviális hibák

- Helyes-e a következő (Java) kód?

```
public static int abs(int num) {
    if (num < 0)
        return -num;
    else
        return num;
}
```

Nemtriviális hibák



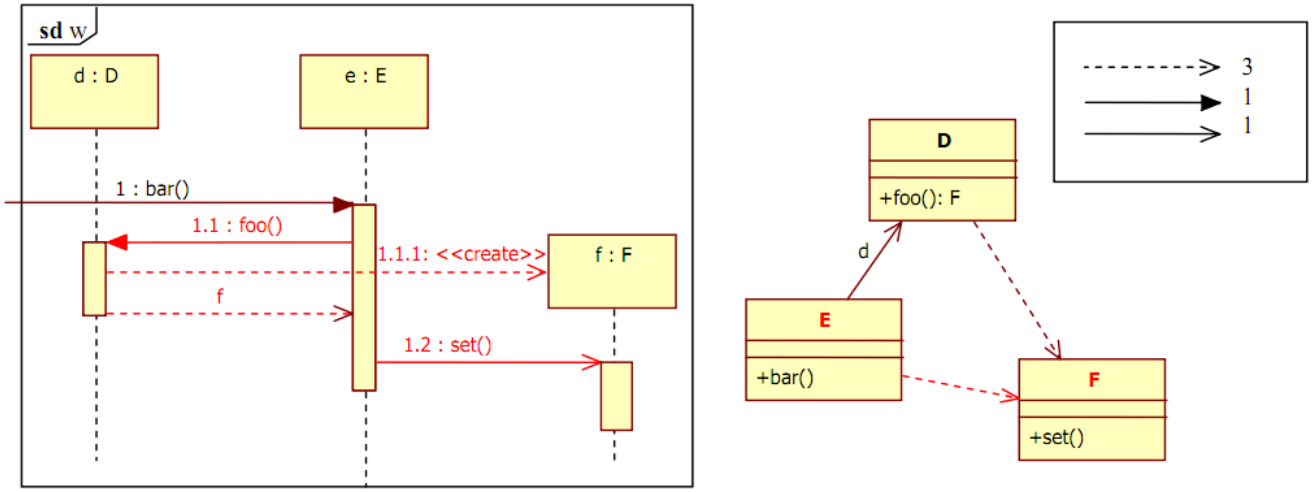


Abs(-2147483648) < 0 ?

Miért nem UML?

- az UML szintaktikája precízen definiált

1.Az ábrán egy UML2 osztálydiagram és egy UML2 szekvenciadiagram látszik. A két diagram szemantikailag összefügg, de hiányos. Rajzolja be a hiányzó jelölő-elemeket! Ahol lehet, lássa el őket feliratokkal is! A felhasználható jelölőelemek és számosságuk a mellékelt keretben látható. (5 pont)



Szoftvertechnológia vizsgafeladat, 2012. forrás: http://iit.bme.hu/~stuser/

- az UML szemantikája (jelentése) nem egyértelmű!
  - Z. Micskei and H. Waeselynck: A survey of UML 2.0 sequence diagrams' semantics, LAAS Technical Report, August 2008. (54 oldal!)

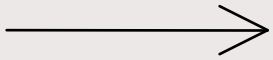
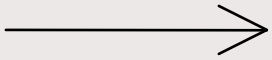
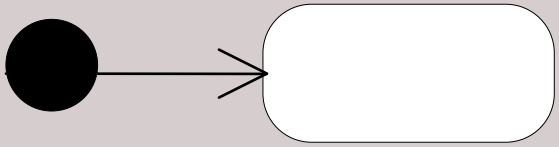
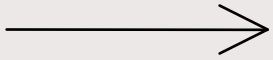
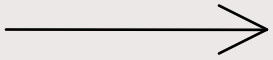
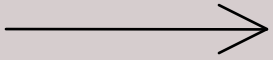
UPPAAL

UPPAAL történet

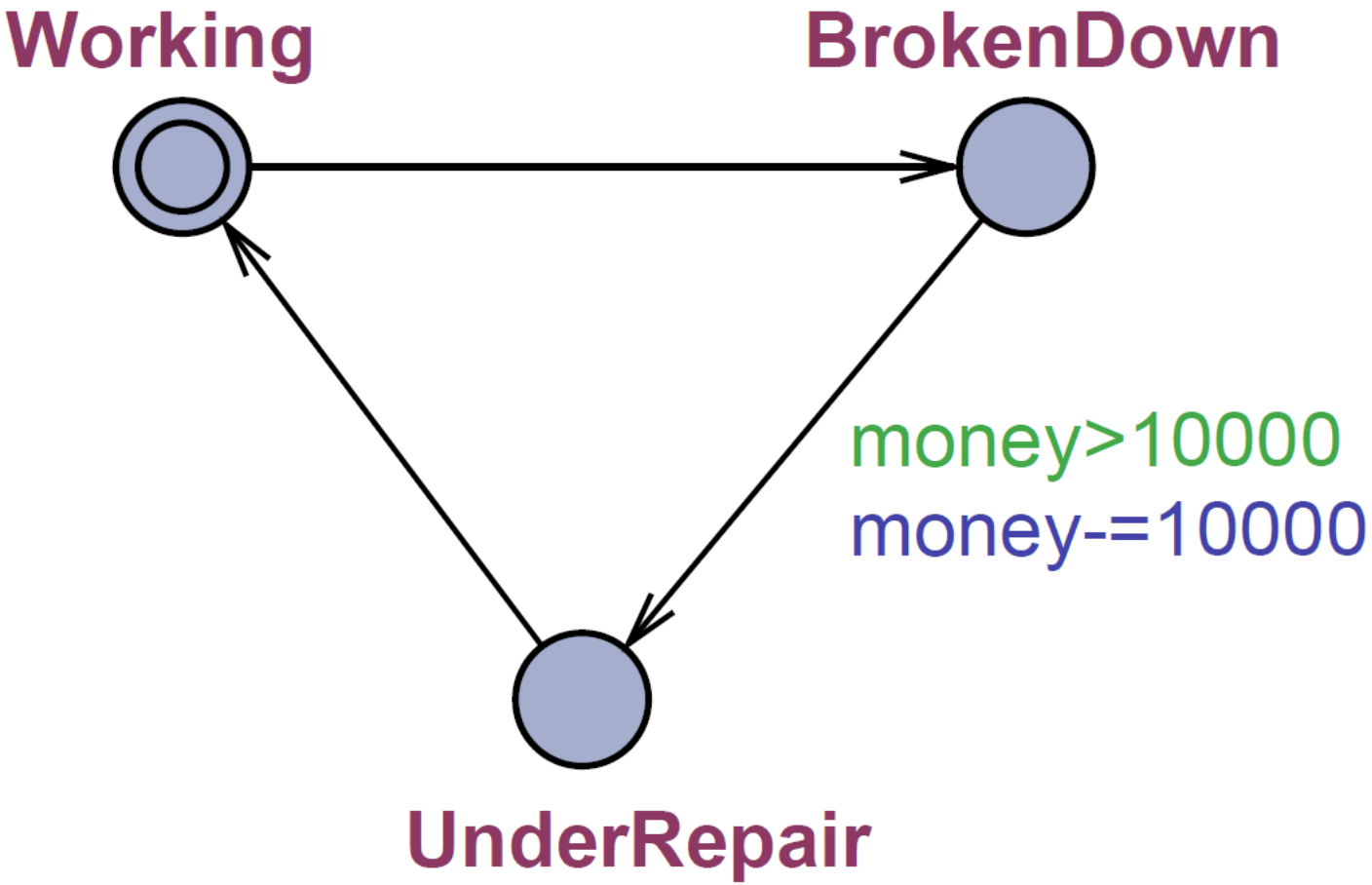
- akadémiai fejlesztés
  (Uppsala University, Aalborg University)
- első megjelent változat: 1995

- **modellező** és **verifikációs** rendszer
  - modellek: *időzített automaták hálózata*
    - a mi céljainkra tekinthetjük ezt *kibővített állapotgépek hálózatának*
    - valójában: állapotgép ⊆ automata
  - verifikáció: CTL részhalmaz (később)

Állapotgép elemei

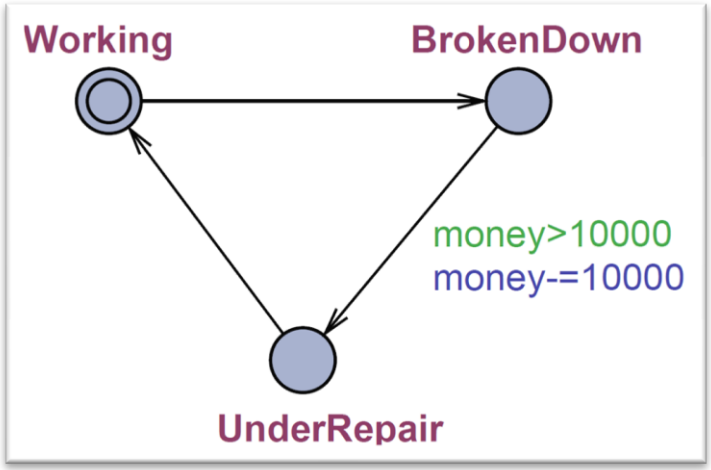
UML State Chart		UPPAAL
	**Állapot**	
	**Állapotátmenet**	
	Kezdőállapot	
[money>1000] 	Állapotátm. feltétele (őrfeltétel)	money>1000 
/ money -= 1000 	Állapotátmenet akciója	money-=1000 

UPPAAL példamodell

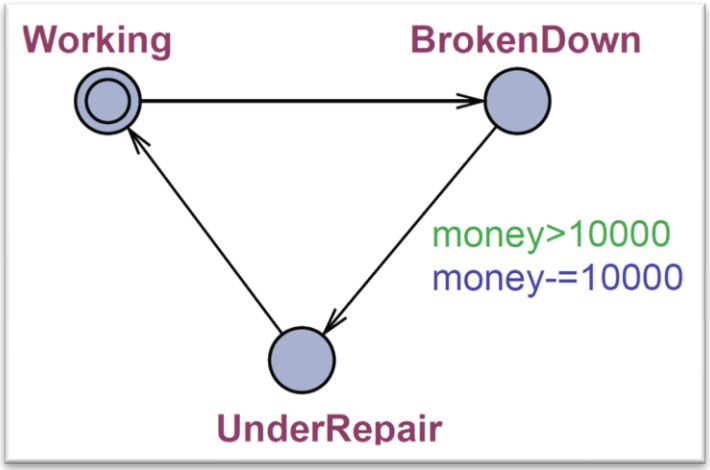


Állapotgépek hálózata

- *„ kibővített állapotgépek **hálózata**”*



System 1

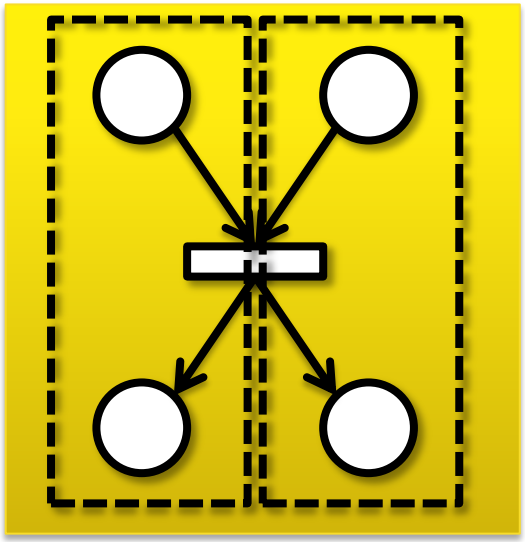


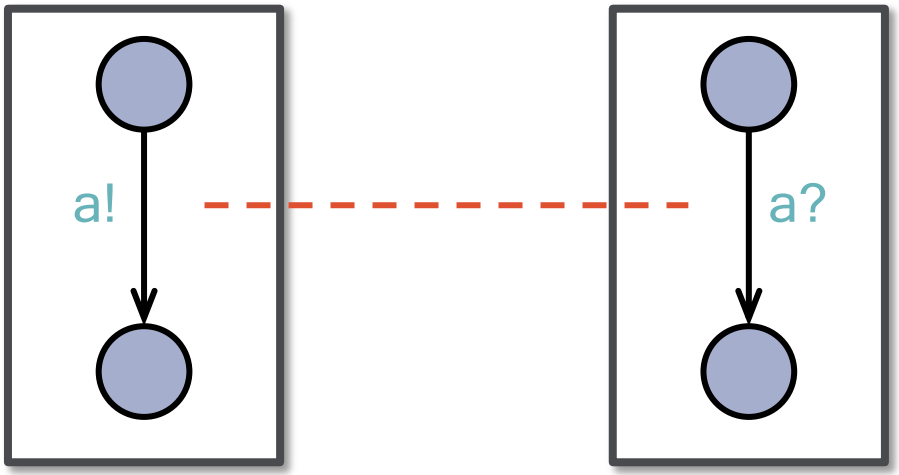
System 2

- a két rendszer különböző állapotban lehet

Állapotgépek hálózata

- a különböző állapotgépek nem függetlenek
- **szinkronizáció** lehetséges köztük (sync)
  - csatornát deklarálni kell: `chan a;`

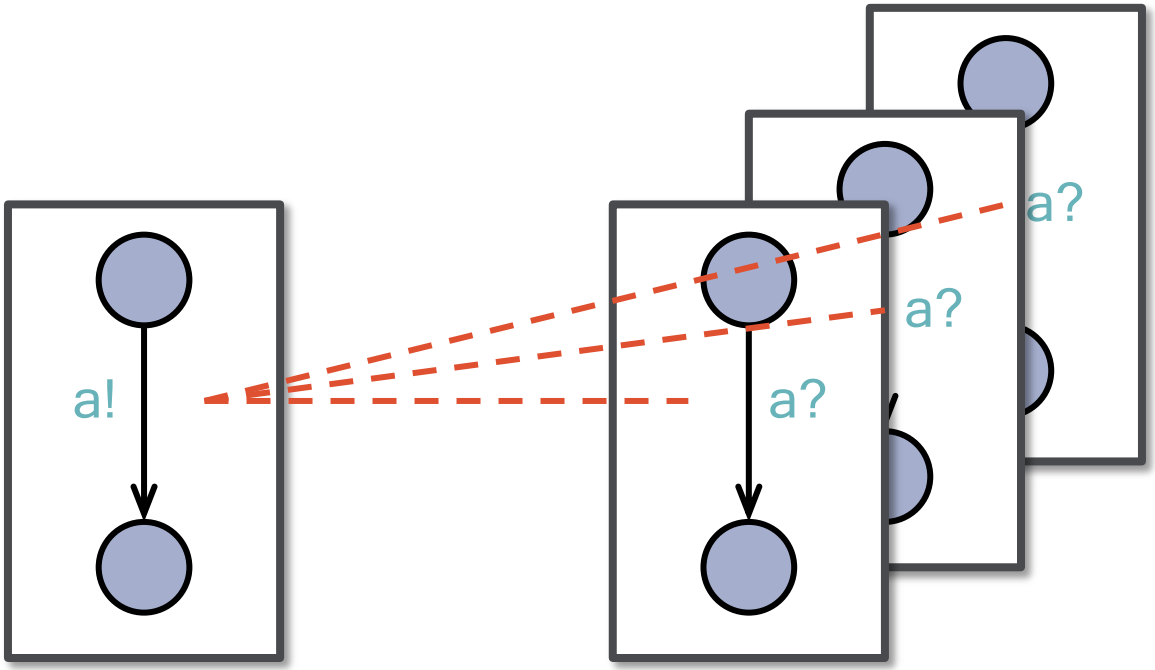




- továbblépés pontosan akkor, ha a csatornán van **küldés** (a!) és **fogadás** is (a?)

Állapotgépek hálózata

- **broadcast szinkronizáció** is lehetséges
  - csatornát deklarálni kell: `broadcast chan a;`
  - működése:



    - a **küldő** (a!) mindig továbbléphet és ilyenkor az összes **fogadó** (a?) is továbbléphet

Terminológia helyrerázása

- **template**: egy állapotgép-típus leírása
- **példány**: egy konkrét állapotgép, amelynek lehet saját [aktuális] állapota
  - egy template-ből több példány is lehet
- **rendszer**: példányok összessége

- valójában *template-eket* rajzoltunk
- példányosítás:
- rendszer definiálása:

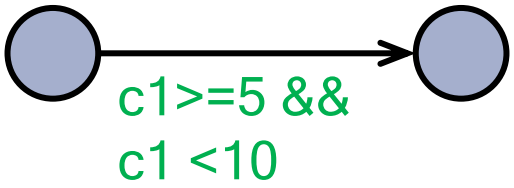
```
i1 = TemplateName();

system i1, i2, i3;
```

System declarations

Óraváltozók

- *„**kibővített** állapotgépek hálózata”*
- kibővítés: pl. időzítés
  - definiálhatók **óraváltozók** (logikai órák):
    `clock c1;`
  - pontos értékét nem ismerjük, csak azt, hogy értéke milyen tartományba esik
  - pl.



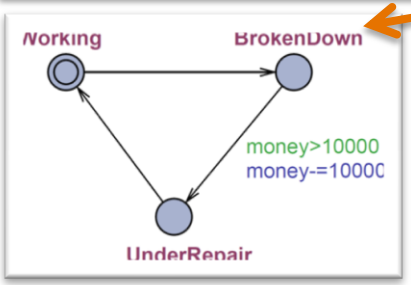
    - itt amikor az állapotátmenet megtörténik, c1∈[5,10), de ez alapján többet nem tudunk
  - (az órát nem nekünk kell léptetni)

Deklarációk

- deklarálnunk kell a változókat, óraváltozókat, csatornákat, példányosításokat, a rendszert

[Global] declarations

```
broadcast chan bc1;
```

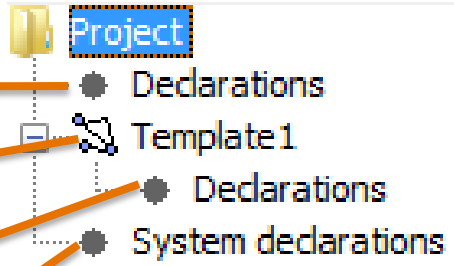


Template1 local declarations

```
int money = 10000;
clock cl1;
```

System declarations

```
i1 = Template1();
system i1, i2, i3;
```

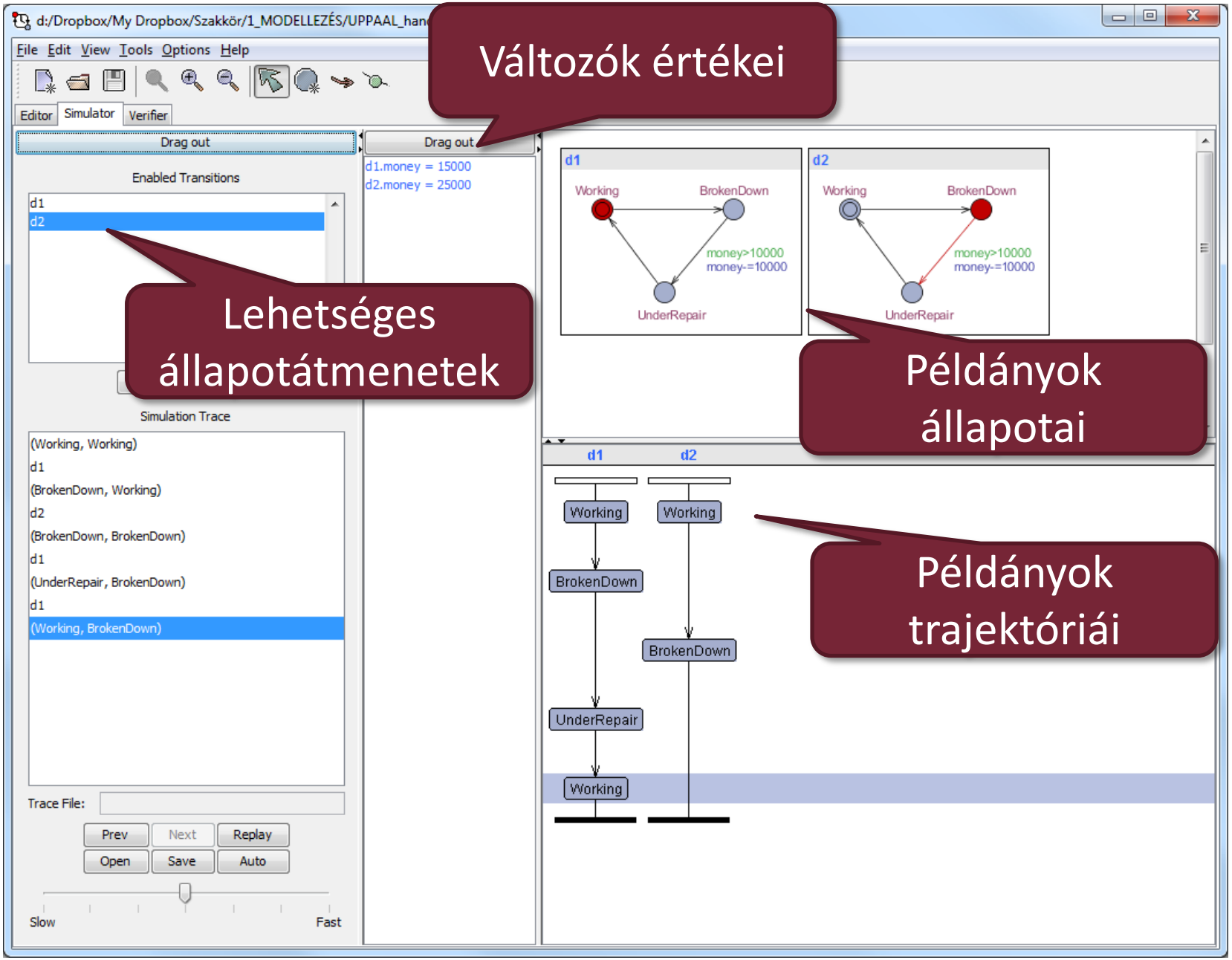


Ami kimaradt…

- adattípusok specifikációja
- őrfeltételek és akciók szintaktikája
- urgent és commited állapotok
- template-ek paraméterezése
- …

- bővebben:
  - http://people.cs.aau.dk/~srba/courses/SV-05/slides/l11.pdf

Szimuláció


Változók értékei
Lehetséges állapotátmenetek
Példányok állapotai
Példányok trajektóriái
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Drag out
d1.money = 15000
d2.money = 25000
Enabled Transitions
d1
d2
Simulation Trace
(Working, Working)
d1
(BrokenDown, Working)
d2
(BrokenDown, BrokenDown)
d1
(UnderRepair, BrokenDown)
d1
(Working, BrokenDown)
Trace File:
Prev
Next
Replay
Open
Save
Auto
Slow
Fast
d1
Working
BrokenDown
money>10000
money-=10000
UnderRepair
d2
Working
BrokenDown
money>10000
money-=10000
UnderRepair
d1
d2
Working
Working
BrokenDown
BrokenDown
UnderRepair
Working

Verifikáció

Verifikáció

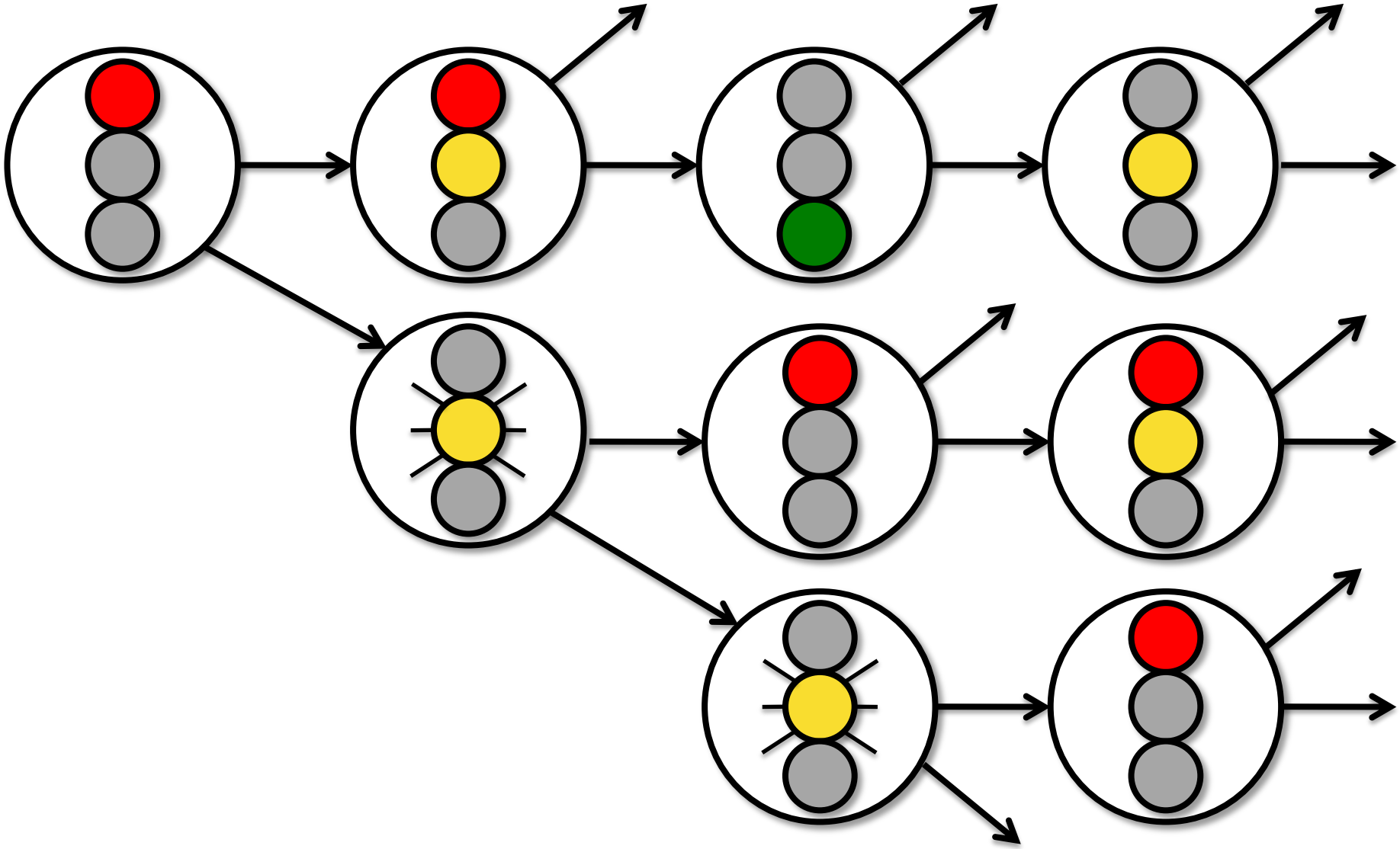
- specifikációknak való megfelelés ellenőrzése
- biztonságkritikus rendszereknél elengedhetetlen

- egy hibás rendszer
  - balesetet okozhat
    - Mi történik, ha lefagy egy repülőgép vezérlője?
  - ronthatja egy termék/szolgáltatás megítélését
    - Szívesen vennél ki pénzt egy X bank ATM-éből, ha többször is kékhalált látnál rajta?
  - a hibajavítás drága lehet
    - Egy mosógépen/autón nincs Windows Update, hibajavítás csak kiszállással vagy visszahívással lehetséges.

Temporális logika dióhéjban

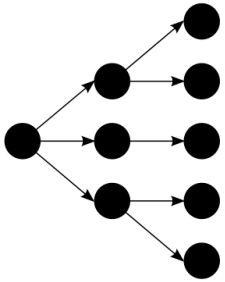
- feltételek időbeli változásait vizsgálhatjuk vele
- UPPAAL-ban a CTL egy résznyelvét használhatjuk erre a célra
- a kifejezéseket az „összes lehetséges állapotátmenet fáján” (*számítási fán*) értelmezzük
  - (pongyola megfogalmazás)
  - a fa csúcsai (globális) állapotok,
    a fa élei (globális)állapotátmenetek

Példa számítási fára

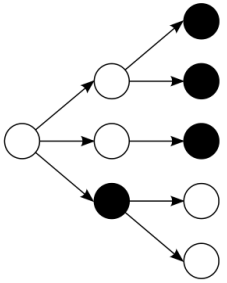


UPPAAL-CTL

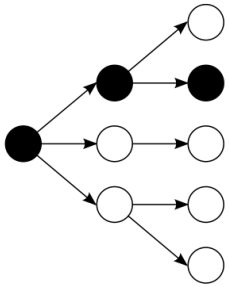
- UPPAAL-ban értelmezett kifejezések:
  - **A[]** *kifejezés*: a fa minden csúcsára (állapotára) igaz



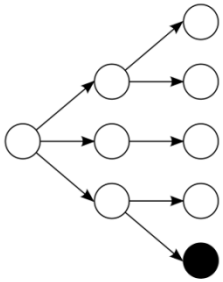
  - **A<>** *kifejezés*: a fa minden útvonalán legalább egy csúcsra igaz



  - **E[]** *kifejezés*: a fa egyik útvonalán minden csúcsra igaz



  - **E<>** *kifejezés*: a fában legalább egy csúcsra igaz



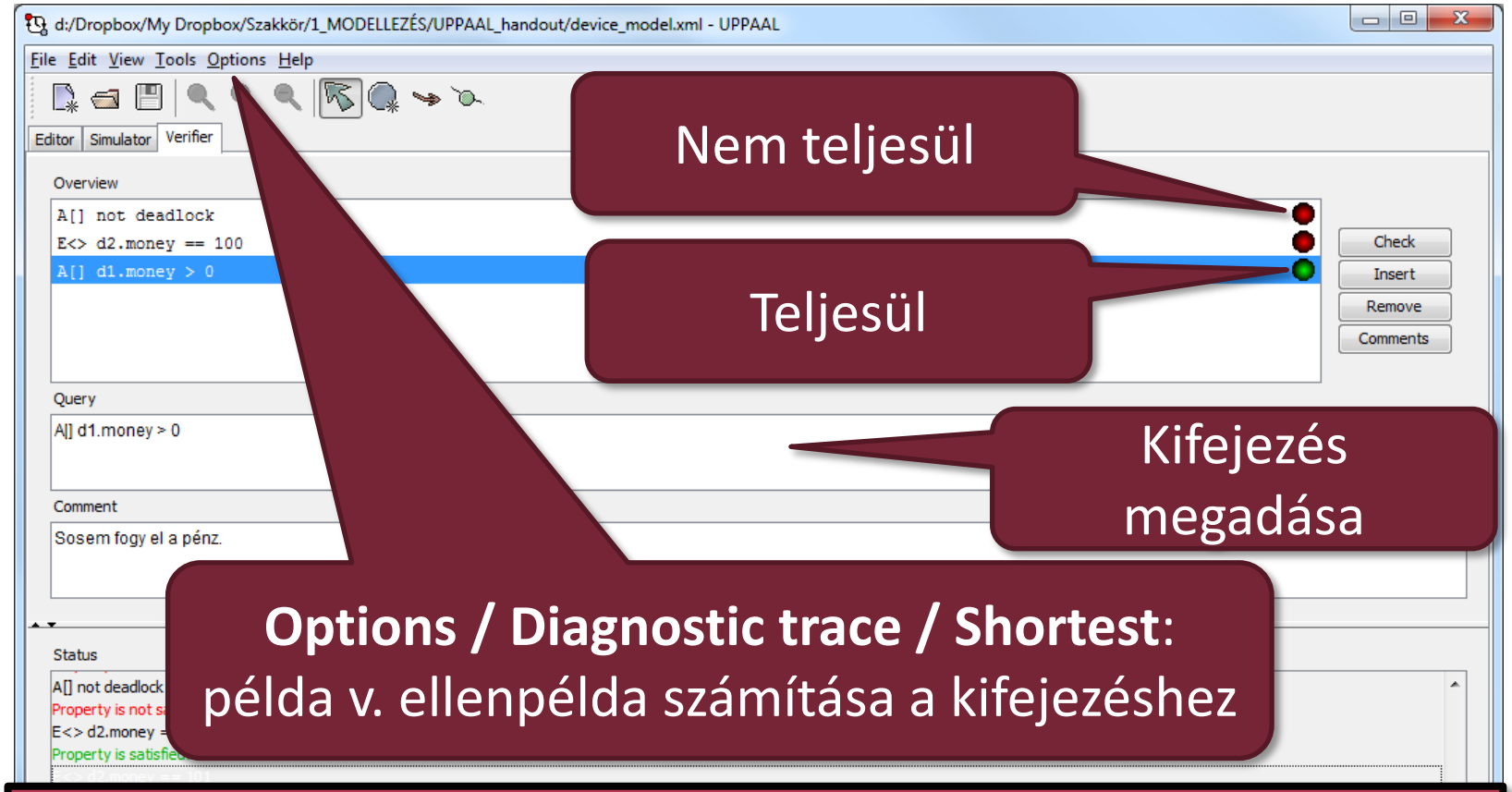
UPPAAL-CTL

- példa kifejezések
  - **`A[] x>=0`**: igaz, ha tetszőleges állapotátmeneteket tetszőleges sorrendben végrehajtva az x változó értéke sohasem lesz negatív
  - **`E<> x>100`**: igaz, ha létezik egy lehetséges állapotátmenet-sorozat, amelyet ha végrehajtunk, eljuthatunk olyan állapotba, melyben x értéke 100-nál nagyobb
  - **`E<> !d1.Working && !d2.Working`**: igaz, ha előfordulhat, hogy mindkét eszköz elromlik (tehát nem Working állapotban lesz)

Összefüggések operátorok közt

- speciális kifejezések:
  - **`A[] not deadlock`**: igaz, ha nincs holtpont a rendszerben, mindig van lehetséges állapotátmenet
  - *kif1* **`-->`** *kif2*: igaz, ha minden úton, amelyen található *kif1*, található *kif1* után *kif2* is
- összefüggések:
  - ¬(**A[]** *kifejezés*) = **E<>** ¬*kifejezés*
  - ¬(**A<>** *kifejezés*) = **E[]** ¬*kifejezés*

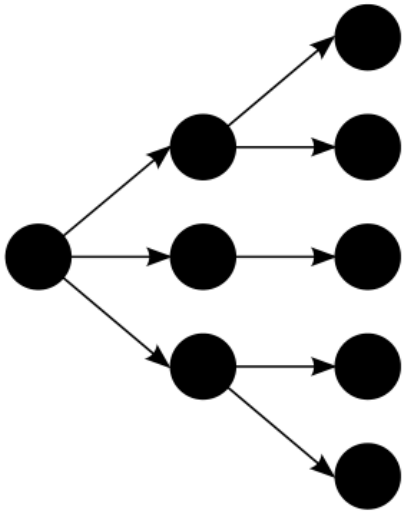
UPPAAL Verifier



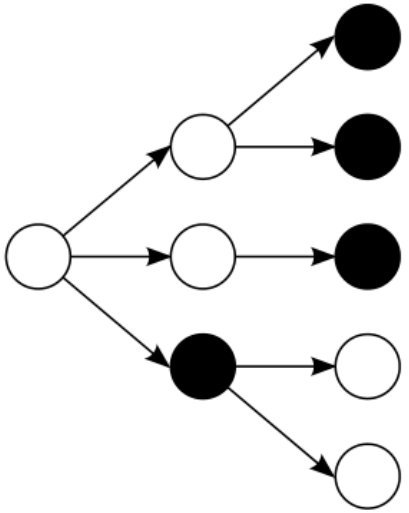
Figyelem!
A Verifierben megadott követelmények nem a modellel együtt kerülnek mentésre, hanem **egy külön .q fájlban**!

UPPAAL CTL operátorok

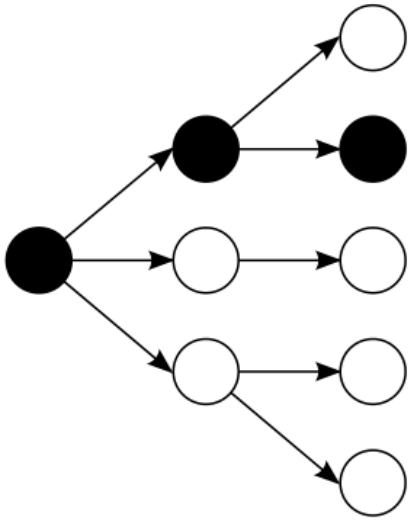
A[]



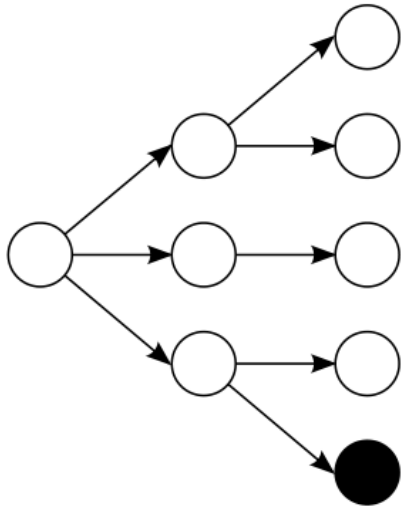
A<>



E[]



E<>



Feladatok

Hyman-algoritmus

Kölcsönös kizárás

- bizonyos erőforrások **csak kizárólagosan használhatók** (pl. egyszerre két program egy nyomtatón ne nyomtasson)
- ennek elkerülésére **kölcsönös kizárási algoritmusok**

Kölcsönös kizárás

- minek erre algoritmusokat írni?

```
boolean locked = false;



if (locked == false) {
  locked = true;

  // do critical stuff

  locked = false;
}
```

Kölcsönös kizárás

```
boolean locked = false;
```

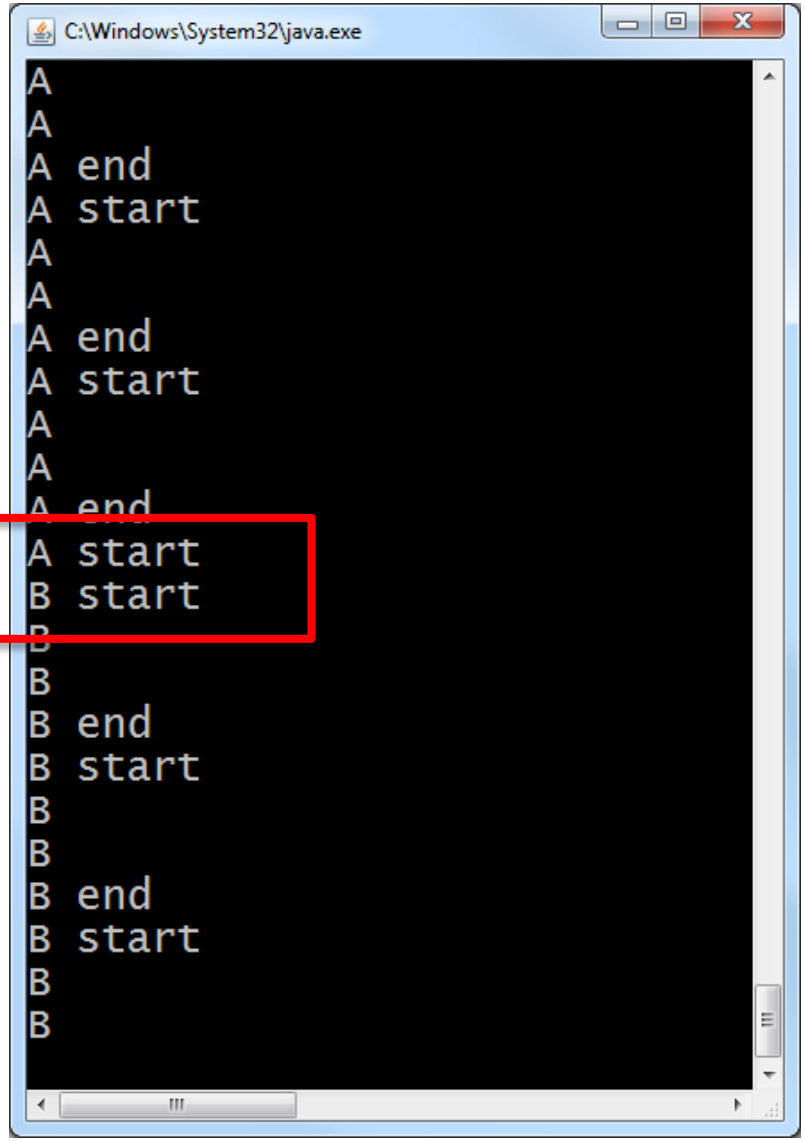
Thread 1:

```
while (true) {
  if (locked == false) {
    locked = true;
    sysout("A start");
    sysout("A");
    sysout("A");
    sysout("A end");
    locked = false;
  }
}
```

Thread 2:

```
while (true) {
  if (locked == false) {
    locked = true;
    sysout("B start");
    sysout("B");
    sysout("B");
    sysout("B end");
    locked = false;
  }
```

Kölcsönös kizárás



A start
B start
B
…

Hyman-algoritmus (1966)

P0 folyamat	P1 folyamat

`while (true) {`
`  blocked0 = true;`
`  while (turn!=0) {`
`  while (blocked1==true) {`
`  skip; //üres ciklus`
`  }`
`  turn=0;`
`  }`
`  // kritikus szakasz helye`
`  blocked0 = false;`
`  // egyéb feladatok`
`}`
	`while (true) {`
	`  blocked1 = true;`
	`  while (turn!=1) {`
	`  while (blocked0==true) {`
	`  skip; //üres ciklus`
	`  }`
	`  turn=1;`
	`  }`
	`  // kritikus szakasz helye`
	`  blocked1 = false;`
	`  // egyéb feladatok`
	`}`

blocked*n*: a P*n* folyamat be akar lépni a kritikus szakaszba

turn: melyik folyamat következhet belépni?

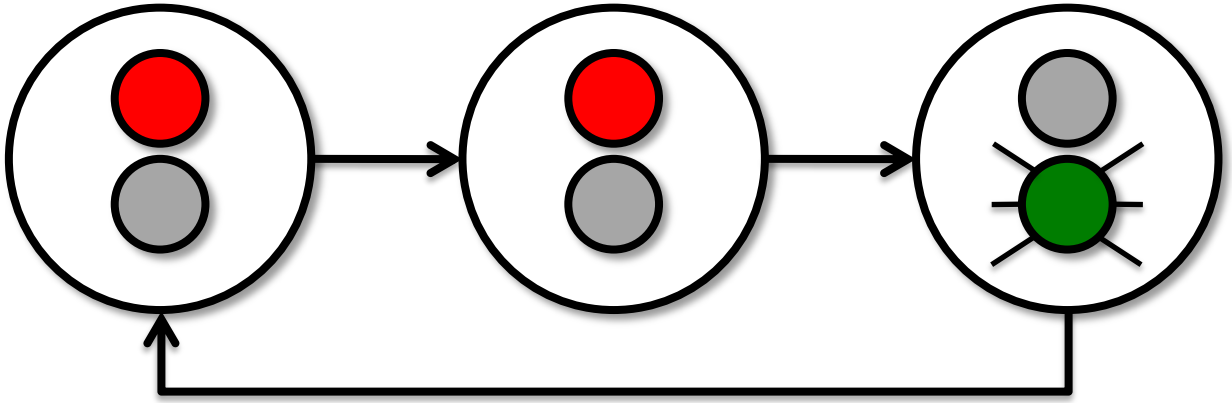
Hyman-algoritmus

- Modellezzük a Hyman-algoritmust két folyamattal!
- Van-e holtpont az algoritmusban?
- Helyes-e a Hyman-algoritmus, azaz kizárja-e azt, hogy egyszerre két folyamat kritikus szakaszban legyen?

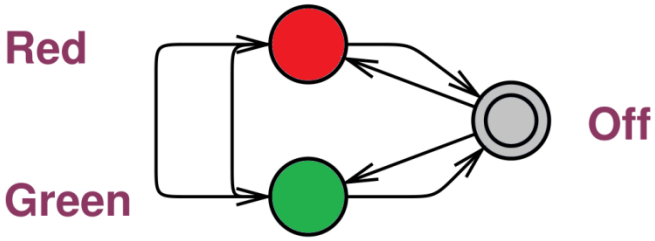
Közlekedési csomópont

Közlekedési csomópont (részfeladat)

- Feladat: egy gyalogos jelzőlámpa modellezése
- állapotok:



- adott a környezeti modell:



lamp3R?
lamp3G?
lamp3R?
lamp3G?
lamp3Off?
lamp3Off?

- **ez még nem vezérlő**: minden állapotból minden állapotba tetszőlegesen át lehet lépni