Követelmények formalizálása: Temporális logikák

dr. Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Motiváció
Mire kellenek a temporális logikák?

Mintapélda: Kölcsönös kizárás

- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

P1

```
while (true) {
    blocked1 = true;
    while (turn!=1) {
        while (blocked0==true) {
            skip;
        }
        turn=1;
    }
    // Critical section
    blocked1 = false;
    // Do other things
}
```

Helyes-e ez az algoritmus?

A modell UPPAAL-ban

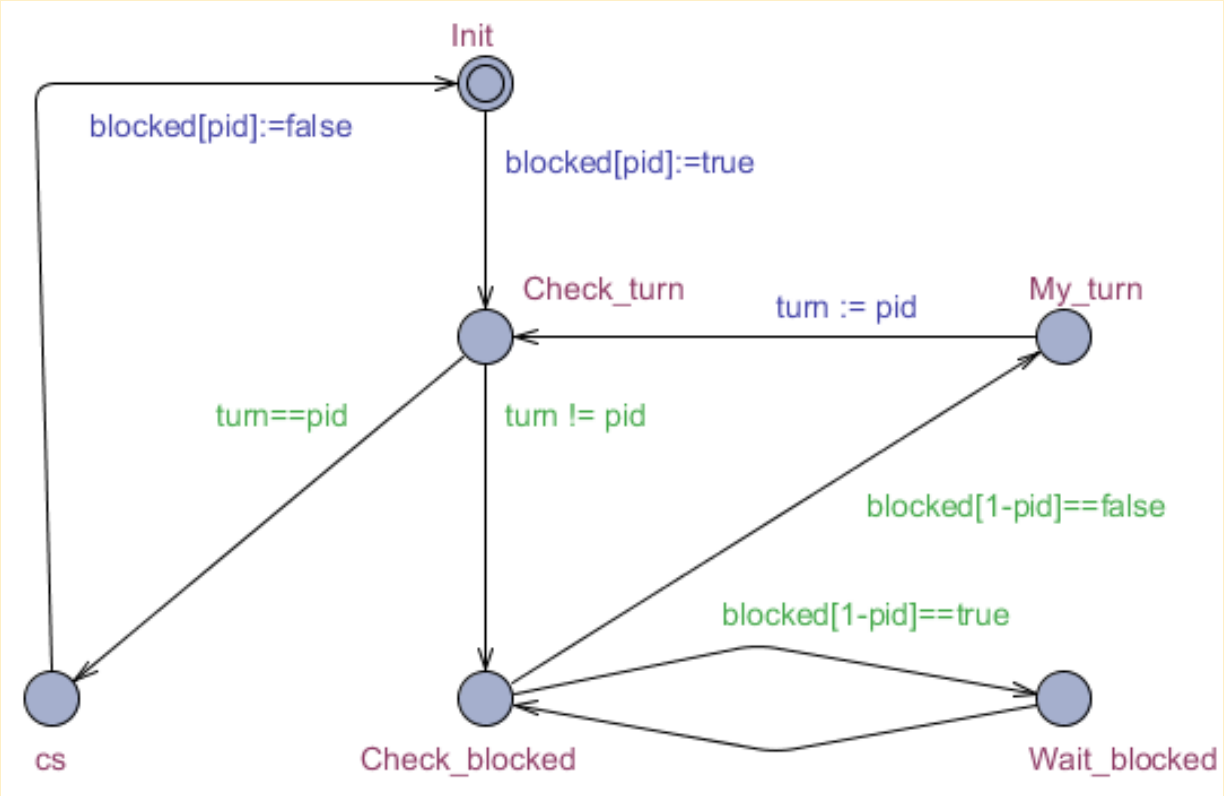
Deklarációk:

```
int[0,1] blocked[2];
int[0,1] turn;
P0 = P(0);
P1 = P(1);
system P0,P1;
```

Kihasznált modellezési lehetőségek:

- Közös változók rendszerszintű deklarálása
- Azonos viselkedésű résztvevők azonos automata template alapján
- Példányosítás paraméterezéssel
- Korlátozott értékkészletű változók
- Változó tömbök (résztvevőkhöz)

A P automata:



P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

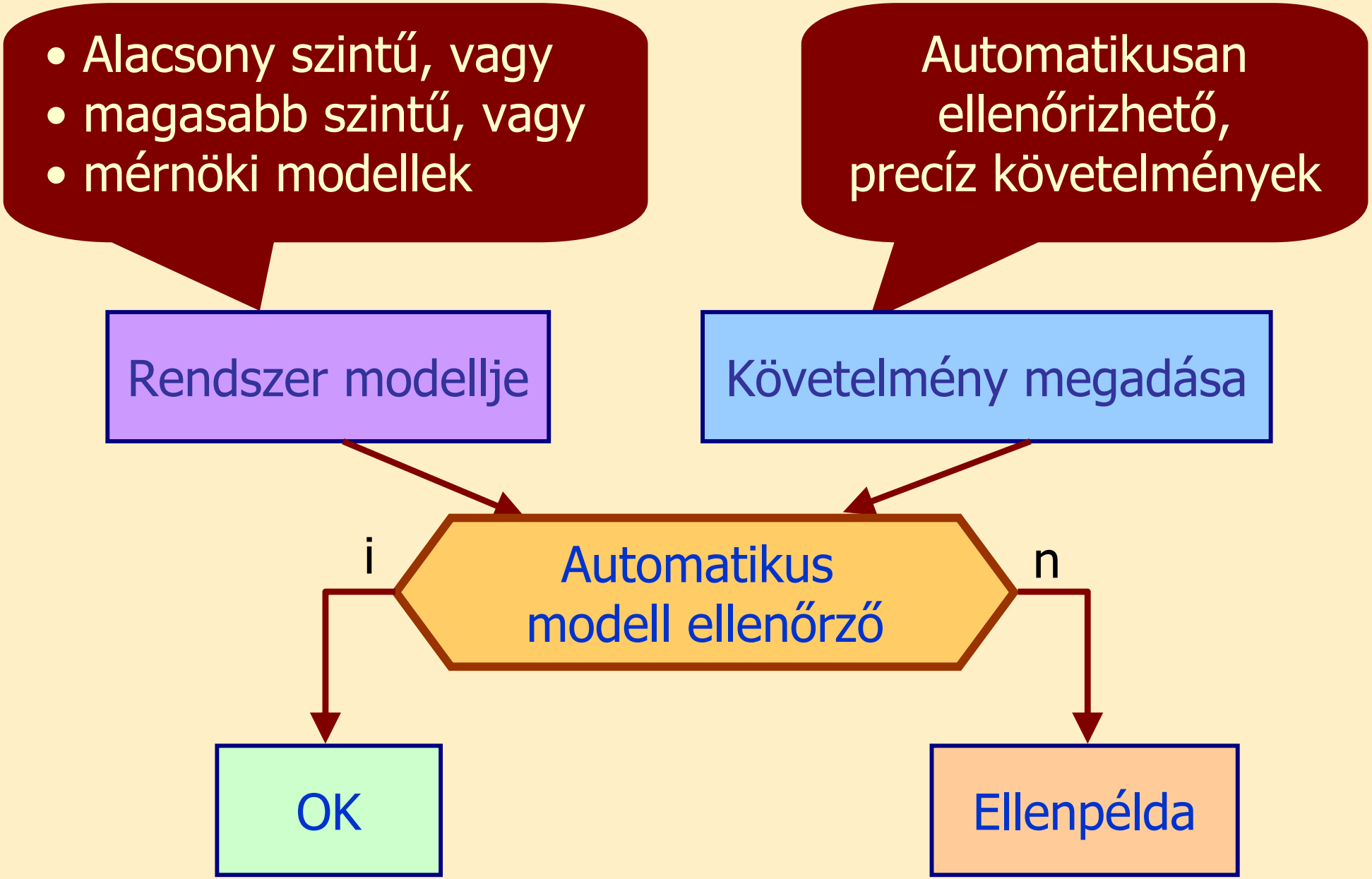
Ellenőrizendő követelmények

- Kölcsönös kizárás:
  - Egyszerre csak az egyik résztvevő lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán beléphet a kritikus szakaszba
  - P1 egyáltalán beléphet a kritikus szakaszba
- Nincs kiéheztetés:
  - P0 mindenképpen be fog lépni a kritikus szakaszba
  - P1 mindenképpen be fog lépni a kritikus szakaszba
- Holtpontmentesség:
  - Nem alakul ki kölcsönös várakozás (leállás)

Hogyan ellenőrizhetjük a követelményeket?

- Teszteléssel
  - Létre tudunk-e hozni minden lehetséges végrehajtást lefedő teszt eseteket?
  - A problémás esetek figyeléséhez külön ellenőrző kell
  - A hiba már csak az implementáció után derülhet ki, drágán javítható
- Modellezéssel és szimulációval
  - Tudunk-e szimulálni minden lehetséges végrehajtást?
  - A problémás esetek detektálása nagy odafigyelést igényel
  - A hibák viszont olcsóbban javíthatók modell szinten
- Modellezéssel és az állapottér teljes ellenőrzésével
  - Szisztematikus algoritmus kell az állapottér teljes felvételéhez
  - Legyen automatikus a követelmények teljesülésének figyelése is
    - Ehhez egy olyan nyelv szükséges, amikkel a követelmények leírhatók
    - Általános módszer adható a követelmények modell alapú ellenőrzésére

Mit szeretnénk elérni?

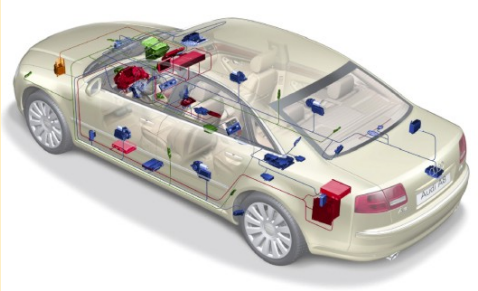


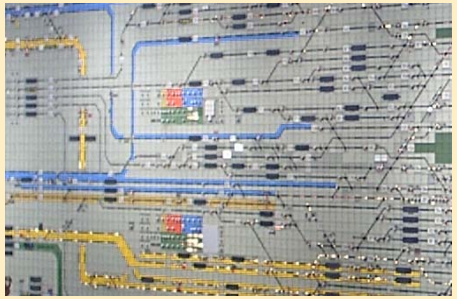
Követelmények formalizálása

Milyen jellegű követelményeket formalizálunk?

- Verifikáció: Modell <-> Sokféle követelmény
  - Funkcionális: Logikailag helyes a működés <- Most ez a célunk
  - Extra-funkcionális: Teljesítmény, megbízhatóság, … is
- Célkitűzés: Állapotok elérhetőségének ellenőrzése
  - Állapotok bekövetkezési sorrendjét vizsgáljuk
    - Eljuthatunk-e kedvező állapotba? -> Élő jellegű követelmények
    - Elkerüljük-e a kedvezőtlen állapotokat? -> Biztonsági követelmények

    Ezek az állapottér teljes felderítésével ellenőrizhetők!
  - Állapotok lokális tulajdonságára hivatkozunk
    - Állapot neve, állapotváltozók értéke
- Fontos állapot alapú, eseményvezérelt rendszerekben





„Biztonsági” jellegű követelmények

- Veszélyes helyzetek elkerülését írják elő:
  - „Minden állapotban kisebb a nyomás a kritikusnál.”
  - „A présgép csak lecsukott ajtó mellett üzemelhet.”
- Univerzális kvantor az elérhető állapotokon:
  - „Minden elérhető állapotban igaz, hogy …”
  - Invariánst fogalmaznak meg
- Ha egy állapotsorozat nem teljesíti:
  nem is egészíthető ki úgy, hogy teljesítse
- Informatikai példák:
  - Holtpontmentesség
  - Kölcsönös kizárás
  - Adatbiztonság (egyik állapotban sincs jogosulatlan hozzáférés)

„Élő” jellegű követelmények

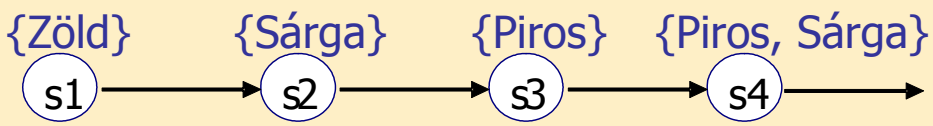
- Kívánatos helyzetek elérését írják elő
  - „Az indítás után a présgép kiadja az elkészült terméket.”
  - „A zavarás után a folyamat visszakerül stabil állapotba.”
- Egzisztenciális kvantor az elérhető állapotokon
  - „Létezik (elérhető) olyan állapot, hogy …”
- Ha egy állapotsorozat nem teljesíti:
  elvileg kiegészíthető úgy, hogy teljesítse
- Informatikai példák:
  - Adott kérés kiszolgálása megtörténik.
  - Elküldött üzenet megérkezik.
  - A processz kiszámítja a várt értéket.

Az elérhetőségi követelmények leíró nyelve

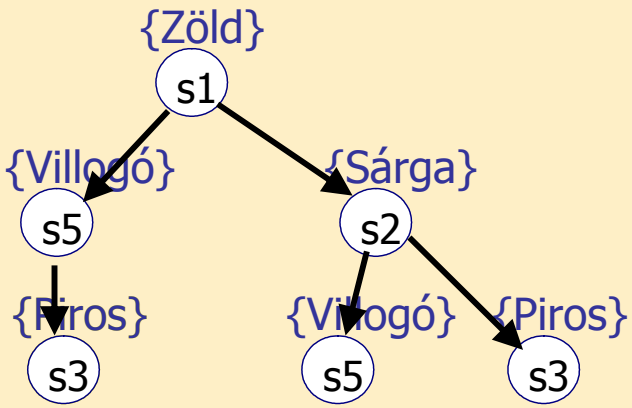
- Elérhetőség:
  Állapotok bekövetkezési sorrendjére vonatkozik
  - Bekövetkezési sorrend: Megfeleltethető a logikai időnek
    - Jelen időpillanat: Aktuális állapot
    - Következő időpillanat(ok): Rákövetkező állapot(ok)
  - Temporális (logikai időbeli, sorrendi) operátorok használhatók a követelmények kifejezésére
- Temporális logikák:
  - Formális rendszer arra, hogy kijelentések igazságának logikai időbeli változását vizsgálhassuk
  - Temporális operátorok: „mindig”, „valamikor”, „mielőtt”, „addig, amíg”, „azelőtt, hogy”, …

Temporális logikák osztályozása

- Lineáris:
  - A modell egy-egy végrehajtását (lefutását) tekintjük
  - Minden állapotnak egy rákövetkezője van
  - Logikai idő mint állapotsorozat jelenik meg



- Elágazó:
  - A modell minden végrehajtását tekintjük
  - Az állapotoknak több rákövetkezője lehetséges
  - Logikai idő mint számítási fa jelenik meg (elágazó idővonalak)



Temporális logikák

Hol értelmezhetjük a temporális logikákat?

- Célkitűzés: Állapottér vizsgálata

Matematikai modell: Kripke-struktúra

- Állapotok lokális tulajdonságait címkézéssel vezettük be

$KS = (S, R, L)$ és AP, ahol

$\mathrm{AP} = \{\mathrm{P,Q,R,...}\}$ atomi kijelentések halmaza (domén-specifikus)

$\mathrm{S} = \{\mathrm{s}_1, \mathrm{s}_2, \mathrm{s}_3, ...\mathrm{s}_\mathrm{n}\}$ állapotok halmaza

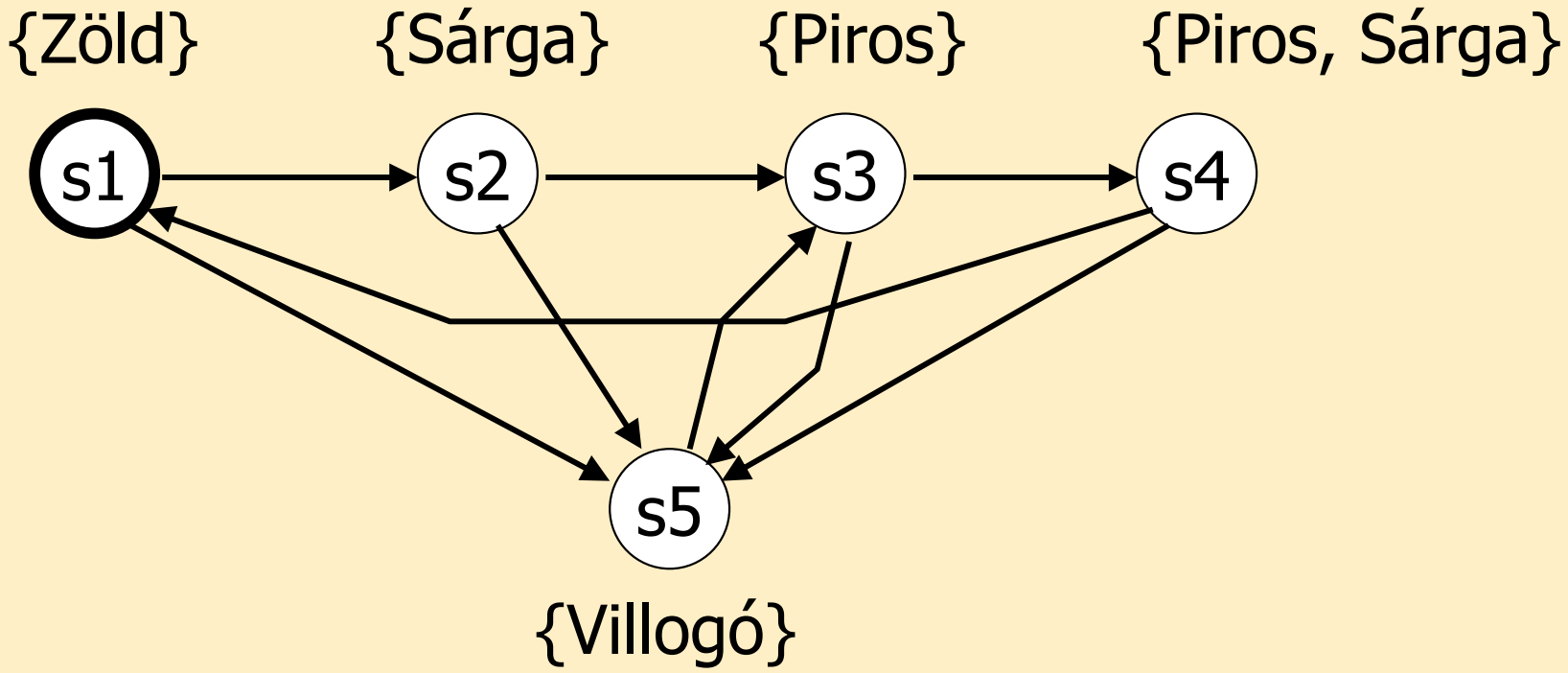
$\mathrm{R} \subseteq \mathrm{S} \times \mathrm{S}$: állapotátmeneti reláció

$\mathrm{L}: \mathrm{S} \rightarrow 2^{AP}$ állapotok címkézése atomi kijelentésekkel

Kripke-struktúra példa

Közlekedési lámpa vezérlője

- AP={Zöld, Sárga, Piros, Villogó}
- S = {s1, s2, s3, s4, s5}

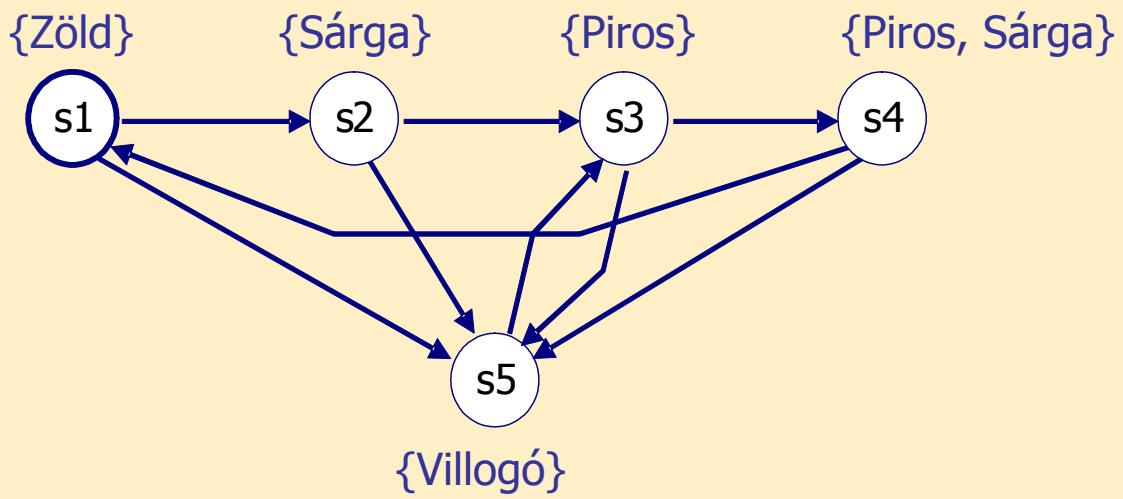


Lineáris idejű temporális logika: PLTL

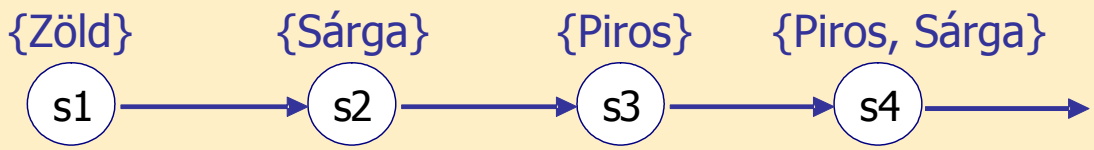
Lineáris idejű temporális logikák

- A Kripke-struktúra egy-egy útvonalán értelmezhetők
  - Egy-egy „lefutás” (pl. egy konkrét bemenet hatására)

A modell (KS):



Egy útvonal (állapotsorozat):



PLTL: Egy lineáris idejű temporális logika

PLTL (Propositional Linear Time Temporal Logic)

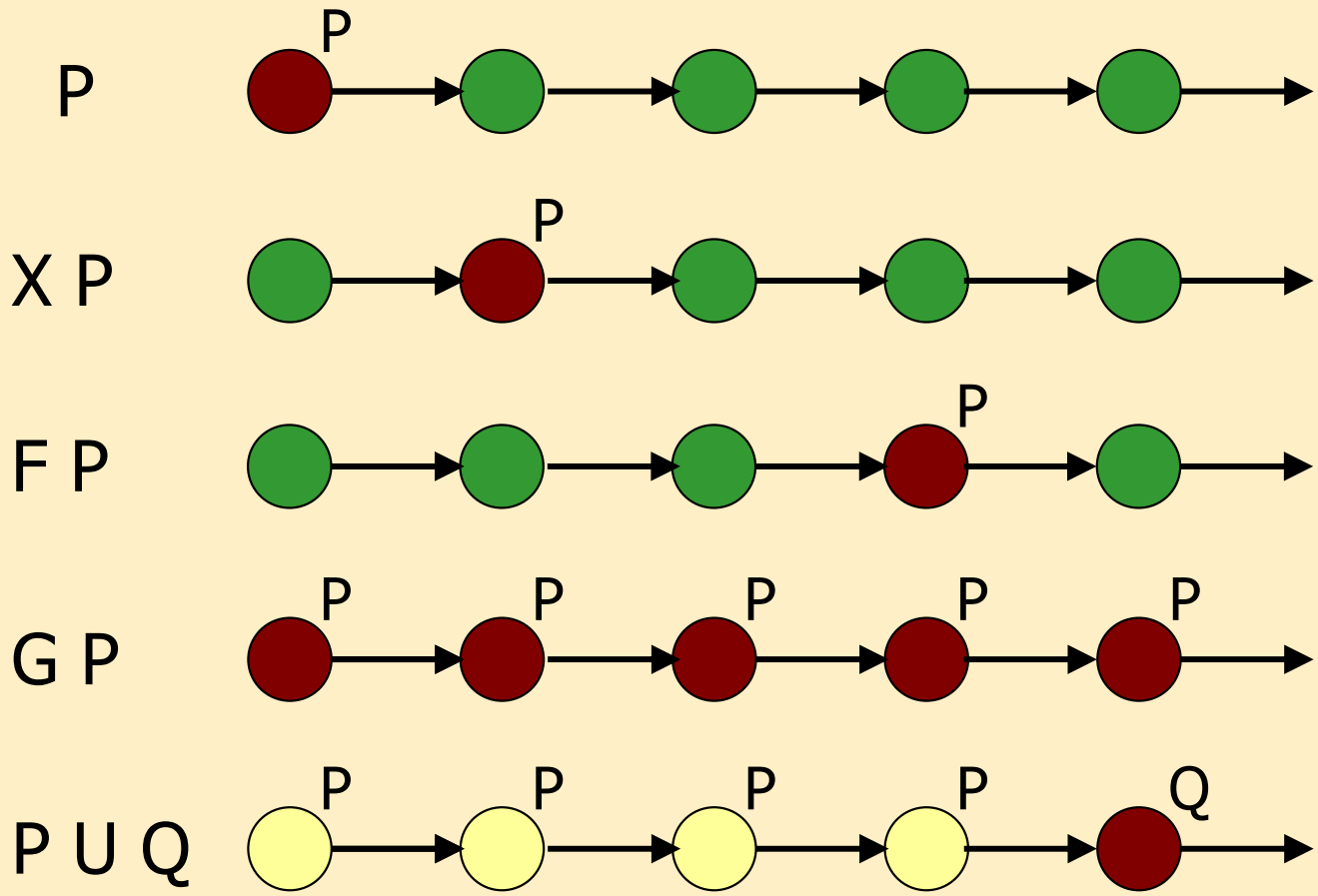
p, q, r, ... kifejezések konstruálása:

- Atomi kijelentések (AP elemei): P, Q, ...
- Boole logikai operátorok: $\wedge$, $\vee$, $\neg$, $\Rightarrow$

  $\wedge$: És, $\vee$: Vagy, $\neg$: Negálás , $\Rightarrow$: Implikáció
- Temporális operátorok: F, G, X, U informálisan:
  - F p: „Future p”, egy elérhető állapotban igaz lesz p
  - G p: „Globally p”, minden elérhető állapotban igaz lesz p
  - X p: „neXt p”, a következő állapotban igaz lesz p
  - p U q: „p Until q”, egy elérhető állapotban igaz lesz q, és addig minden állapotban igaz p

PLTL temporális operátorok

Kripke-struktúra egy útvonala (idővonal):



PLTL példák I.

- $p \Rightarrow Fq$

  Ha a kiindulási állapotra p igaz, akkor valamikor (egy későbbi állapotra) q is igaz lesz.

- $G(p \Rightarrow Fq)$

  Minden állapotra fennáll,
  hogy ha p igaz, akkor valamikor q is igaz lesz.
  Pl. bármikor kiadott p kérésre q válasz érkezik.

- $p\ U\ (q \vee r)$

  Itt p igaz, amíg q vagy r igaz nem lesz.
  Pl. az első állapotból kiadott folyamatos p kérést q válasz vagy r elutasítás követ.

- $(p \wedge G(p \Rightarrow Xp)) \Rightarrow Gp$

  A matematikai indukció leírása: Mindig teljesül

PLTL példák II.

- GF p

  Minden állapotra igaz, hogy ebből indulva valamikor p igaz lesz.

  - Nem találunk olyan állapotot, ami után p tulajdonságú állapot ne lenne elérhető.
  - Pl. minden állapotból a p tulajdonságú kezdőállapotba vihető a rendszer.

- FG p

  Valamikor olyan állapotba kerül a rendszer, hogy azontúl p folyamatosan igaz lesz.

  - Pl. kezdeti tranziens után a p tulajdonságú üzemi állapotokba kerül a rendszer.

PLTL nyelv formális kezelése

- Az eddigiek csak informális bevezetést adtak
  Kérdések vetődhetnek fel:
  - F p igaz-e, ha p rögtön az első állapotban igaz?
  - p U q igaz-e, ha q az első állapotban igaz?
- Az automatikus ellenőrzést is lehetővé tevő precíz megadáshoz szükséges:
  - Formális szintaxis szabályok:
    Mik az érvényes PLTL kifejezések?
  - Formális szemantika szabályok:
    Adott modellen mikor igaz egy PLTL kifejezés?

PLTL formális szintaxis

Az érvényes PLTL kifejezések halmaza a következő szabályokkal képezhető:

- **L1**: Minden P atomi kijelentés egy kifejezés.
- **L2**: Ha p és q egy-egy kifejezés,
  akkor $p \wedge q$ illetve $\neg p$ is
- **L3**: Ha p és q egy-egy kifejezés,
  akkor p U q illetve X p is

Operátorok precedenciája növekvő sorrendben:

$\equiv, \Rightarrow, \vee, \wedge, \neg$, (X,U)

„Kimaradt” operátorok

- true minden állapotra igaz („beépített”)
  false egy állapotra sem igaz
- $p \vee q$ jelentése $\neg((\neg p)\wedge(\neg q))$
  $p \Rightarrow q$ jelentése $(\neg p) \vee q$
  $p \equiv q$ jelentése $(p\Rightarrow q)\wedge(q\Rightarrow p)$
- F p jelentése true U p
  G p jelentése $\neg F(\neg p)$
- „Mielőtt” operátor (before):
  p WB q = $\neg((\neg p)$ U q) (weak before)
  p B q = $\neg((\neg p)$ U q) $\wedge$ F q (strong before)

Informálisan:
Nem igaz, hogy nem fordul elő p a q előtt

PLTL szemantika: Jelölések

- $M=(S, R, L)$ Kripke-struktúra

- $\pi=(s_0, s_1, s_2,\ldots)$ az M egy útvonala, ahol s_0 a kezdőállapot és $\forall i\geq 0$: $(s_i, s_{i+1})\in R$
  - $\pi^i=(s_i, s_{i+1}, s_{i+2},\ldots)$ a π útvonal szuffixe i-től

- $M,\pi \models p$ jelentése:
  az M modellben a π útvonalon igaz p

PLTL szemantika

A szintaxis szabályok alapján képzett kifejezésekhez megadható a formális szemantika:

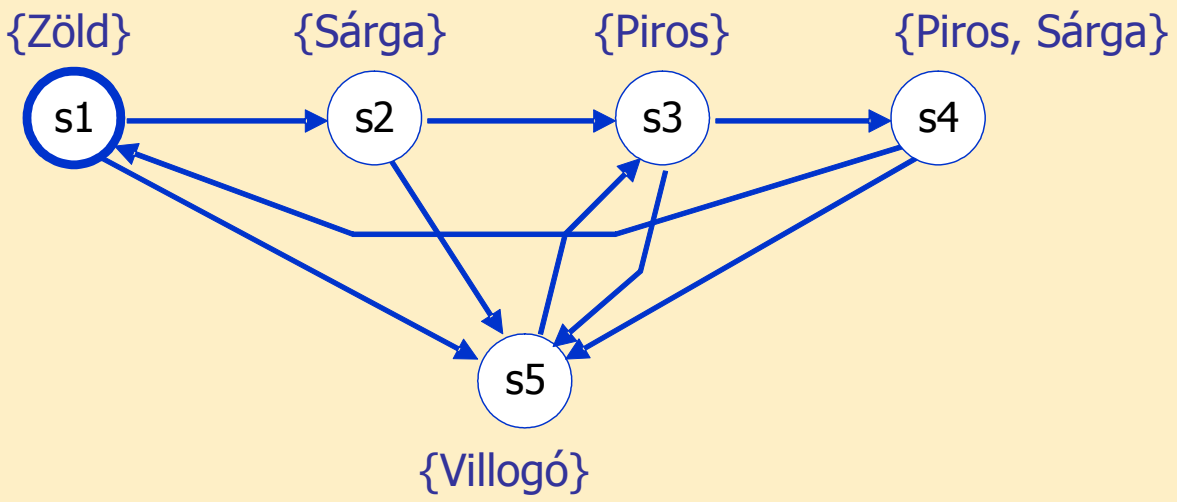
- **L1**: $M,\pi \models P$ a.cs.a. $P \in L(s_0)$
- **L2**: $M,\pi \models p \wedge q$ a.cs.a. $M,\pi \models p$ és $M,\pi \models q$
  $M,\pi \models \neg q$ a.cs.a. $M,\pi \models q$ nem igaz.
- **L3**: $M,\pi \models (p\ U\ q)$ a.cs.a.

$$\exists j \geq 0 : (\pi^j \models q \text{ valamint } \forall 0 \leq k < j : \pi^k \models p)$$

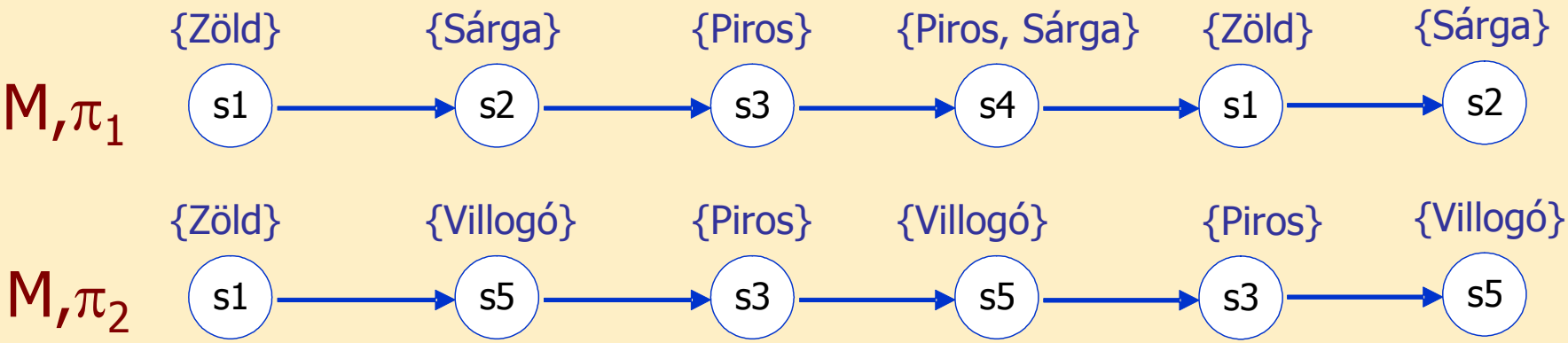
$M,\pi \models X\ p$ a.cs.a. $\pi^1 \models p$

PLTL kifejezések értelmezése: Példák

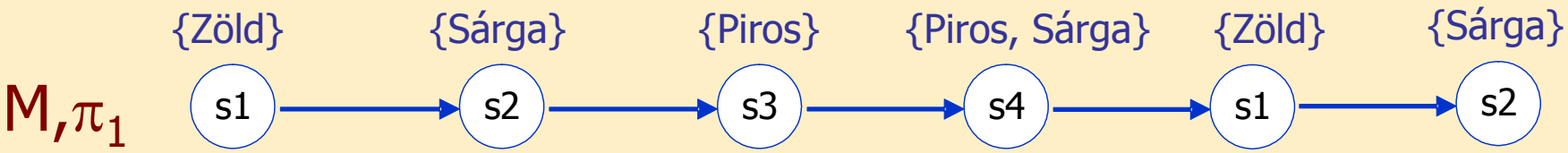
- M Kripke-struktúra:



- Útvonalak:

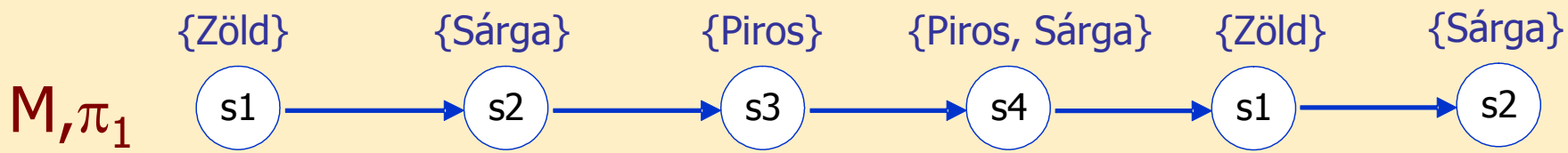


Példák (folytatás)

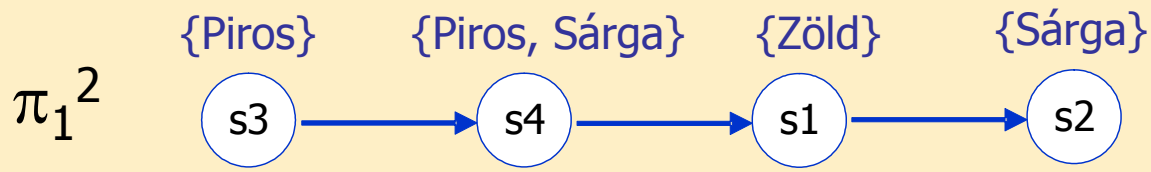


- M,π_1 |= Zöld, mert Zöld a kezdőállapot címkéje
- Piros nem igaz M,π_1 esetén,
  mert nem Piros a kezdőállapot címkéje
- M,π_1 |= F Piros,
  mert elérhető olyan állapot, ahol Piros a címke,
  Pontosabban: Van olyan szuffix, amely esetén
  Piros a kezdőállapot címkéje.
- Zöld U Piros nem igaz M,π_1 esetén,
  mert a Sárga közbeszól

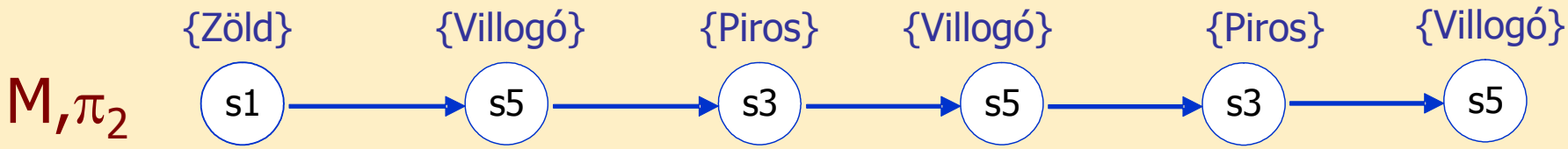
Példák (folytatás)



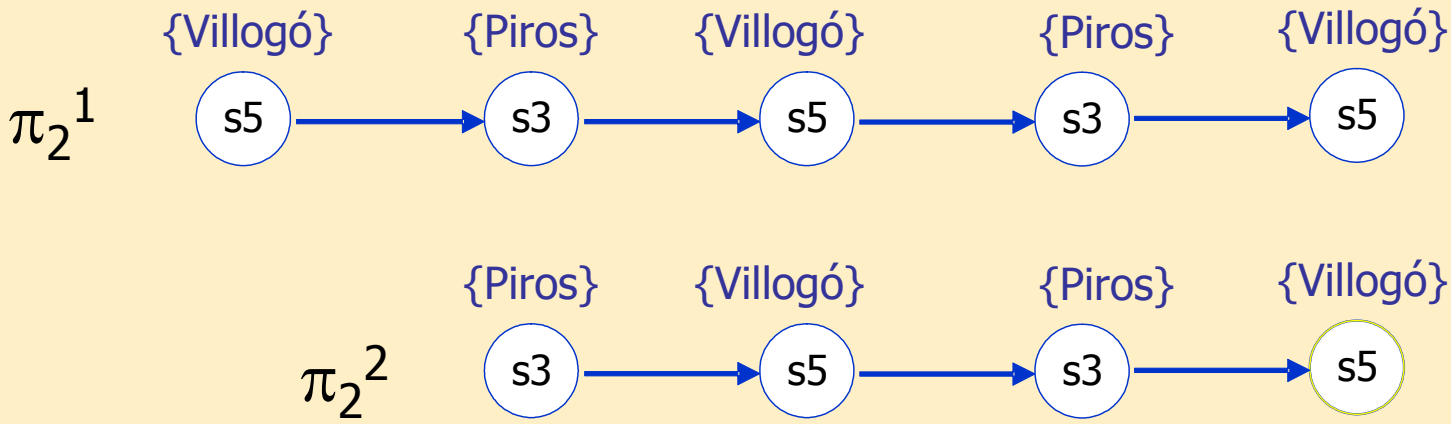
- $M,\pi_1 \models F$ (Piros U Zöld) igaz,
  mert van olyan szuffix,
  amire teljesül a (Piros U Zöld):



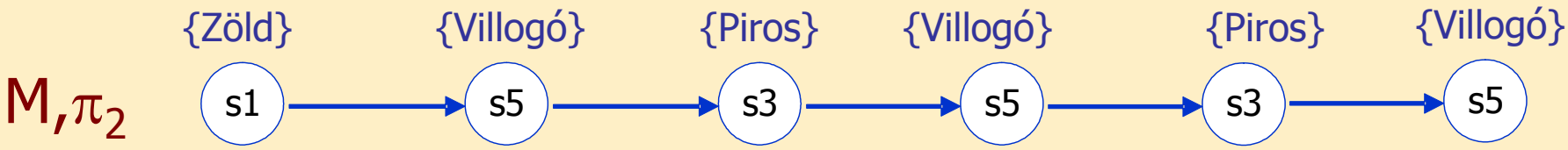
Példák (folytatás)



- $M,\pi_2 \models F$ (Villogó $\Rightarrow$ X Piros),
  mert van olyan szuffix, hogy Villogó $\Rightarrow$ X Piros igaz



Példák (folytatás)



- $M,\pi_2 \models X\ F\ (XX\ Piros)$, mert az első szuffixre:

F (XX Piros) teljesül, mert
π_2^1-nek van olyan szuffixe, ahol XX Piros igaz:

Követelmények formalizálása: Példa

Adott egy klímaberendezés, aminek a következő üzemmódokat kell biztosítania:

AP={Kikapcsolva, Bekapcsolva, Elromlott, GyengénHűt, ErősenHűt, Fűt, Szellőztet}

- Egy-egy állapothoz több címke tartozhat!
- A követelmény formalizálás fázisában a teljes viselkedést még nem ismerjük (azt a követelményeknek megfelelően kell majd megterveznünk)

Példa (folytatás)

AP={Kikapcsolva, Bekapcsolva, Elromlott, GyengénHűt, ErősenHűt, Fűt, Szellőztet}

- A klímát be fogják kapcsolni:
  F (Bekapcsolva)
- A klíma előbb-utóbb mindig elromlik:
  G F (Elromlott)
- Ha a klíma elromlik, mindig megjavítják:
  G (Elromlott $\Rightarrow$ F ($\neg$Elromlott))
- Ha a klíma elromlott, nem fűthet:
  G ($\neg$(Elromlott $\wedge$ Fűt))

Példa (folytatás)

AP={Kikapcsolva, Bekapcsolva, Elromlott, GyengénHűt, ErősenHűt, Fűt, Szellőztet}

- A klíma csak úgy romolhat el, ha be volt kapcsolva:
  G (X Elromlott $\Rightarrow$ Bekapcsolva)
- A fűtési fázis befejezése után szellőztetni kell:
  G ((Fűt $\wedge$ X($\neg$Fűt)) $\Rightarrow$ X (Szellőztet))
  és ha elromlott?
  G ((Fűt $\wedge$ X($\neg$Fűt)) $\Rightarrow$ X (Szellőztet $\vee$ Elromlott))
- Szellőztetés után mindaddig nem hűthet erősen, míg egy gyenge hűtéssel nem próbálkozott:
  G (Szellőztet $\Rightarrow$ ($\neg$ErősenHűt U GyengénHűt))

PLTL modell kiterjesztése LTS-re

- LTS, Labeled Transition System
- Állapotátmenetek címkézhetők egy-egy ún. akcióval, egy átmeneten csak egy akció szerepelhet
- Állapotátmenetek tulajdonságait fejezzük ki

$LTS = (S, Act, \rightarrow)$, ahol

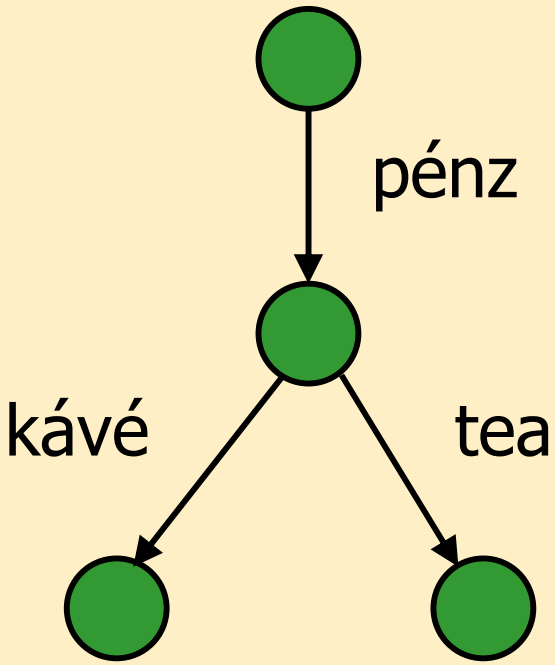
$S = \{s_1, s_2, \ldots s_n\}$ állapotok halmaza

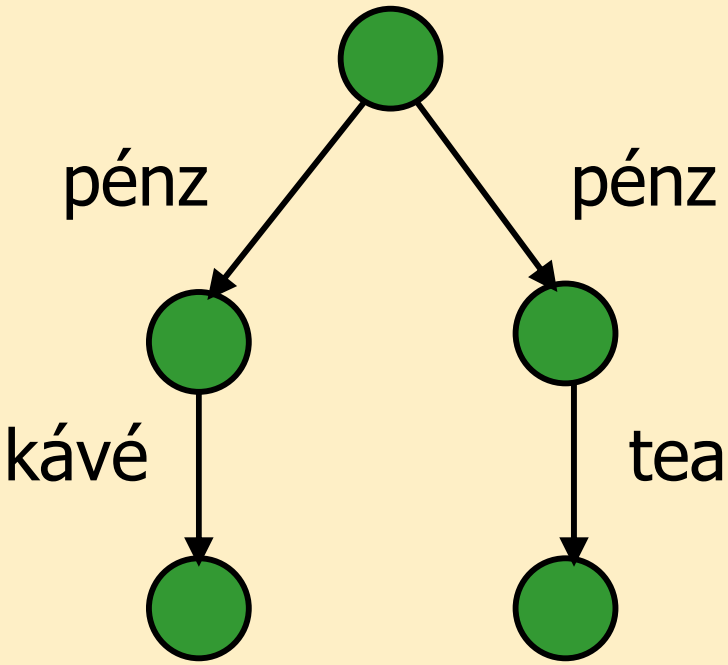
$Act = \{a, b, c, \ldots\}$ akciók (címkék) halmaza

$\rightarrow \subseteq S \times Act \times S$ címkézett állapotátmenetek

Állapotátmenetek szokásos jelölése: $s_1 \xrightarrow{a} s_2$

LTS példák





PLTL értelmezése LTS-en

A struktúra bővülése:

- $\pi = (s_0, a_1, s_1, a_2, s_2, a_3, \ldots)$

A szintaxis módosítása:

- **L1***: Ha a egy akció, akkor (a) egy PLTL kifejezés.

A kapcsolódó szemantikai szabály:

- **L1***: $M,\pi \models (a)$ a.cs.a. $a_1=a$
  ahol a_1 az első akció π-ben.

Ilyen módon üzenetküldéssel kommunikáló rendszerek tulajdonságai is megfogalmazhatók.

PLTL összefoglalás

- Követelmények megfogalmazása
- Temporális logikák
- Lineáris idejű temporális logikák
- PLTL
  - Operátorok (intuitív)
  - Formális szintaxis
  - Formális szemantika
- PLTL kifejezések értelmezése
- Követelmények formalizálása

Elágazó idejű temporális logikák: CTL, CTL*

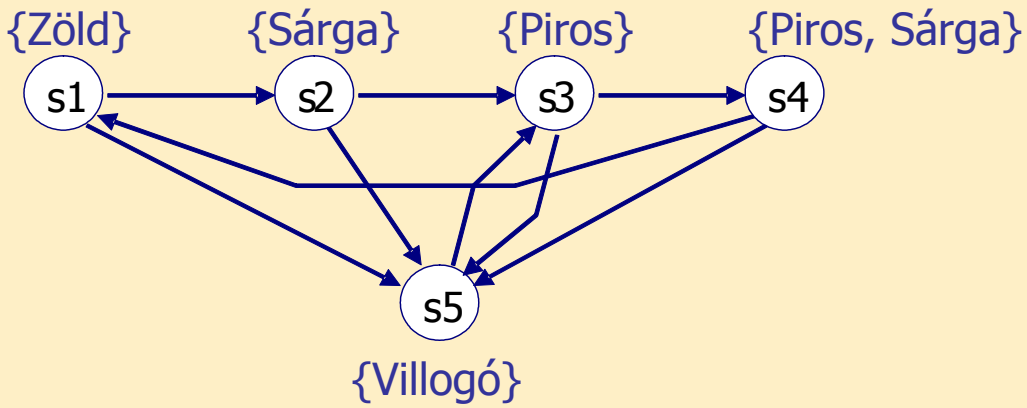
Elágazó idejű temporális logikák

- Lineáris idő: Egy végrehajtás („futás”) ábrázolása
  - Egymás utáni állapotok egy sorozatot alkotnak (minden állapotnak egy rákövetkezője van)
  - Logikai idő mint idővonal ábrázolható
  - Modell ellenőrzés során implicit „minden futásra igaz” jellegű vizsgálatot végzünk

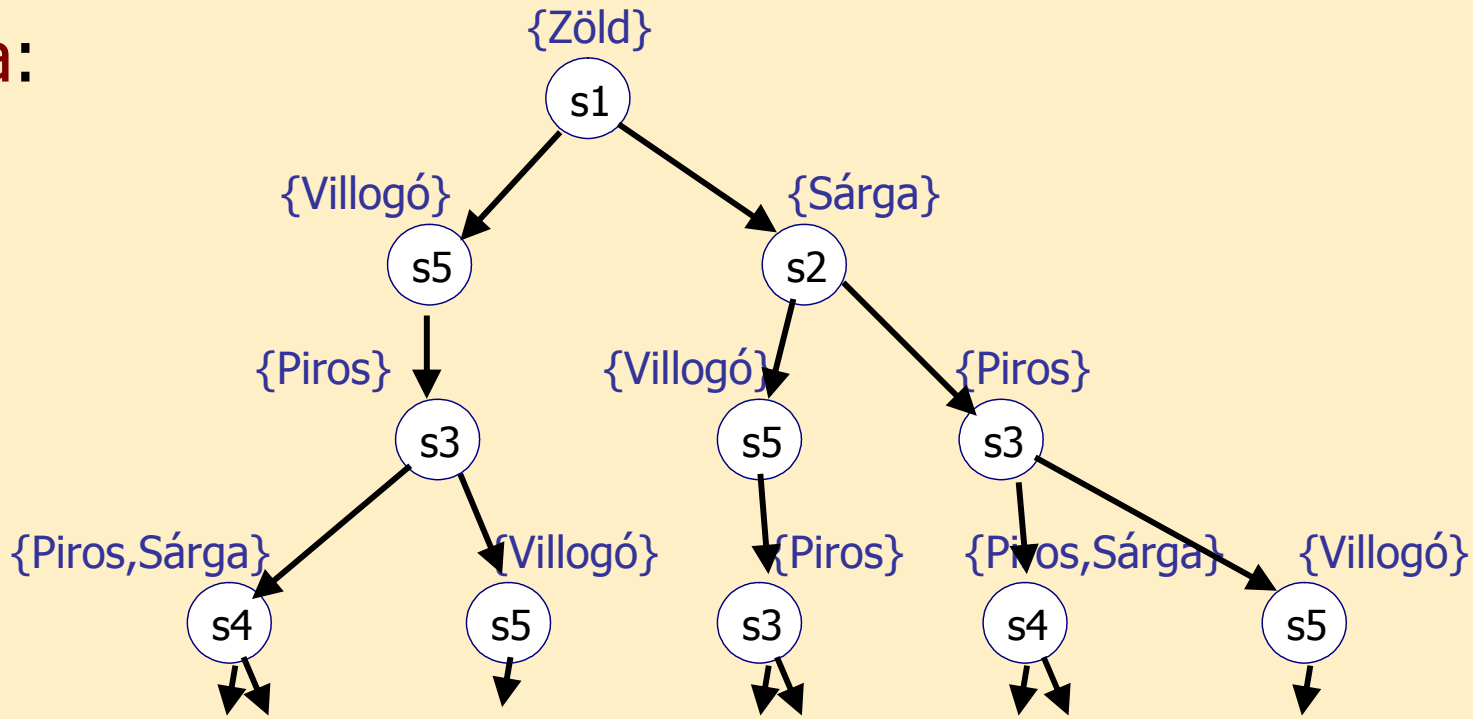
- Elágazó idő: Az elágazások ábrázolása
  - Egymás utáni állapotok egy fa struktúrát képeznek (minden állapotnak több rákövetkezője lehet)
  - Logikai idő elágazó idővonalak mentén ábrázolható

Számítási fa konstrukciója

Kripke-struktúra:



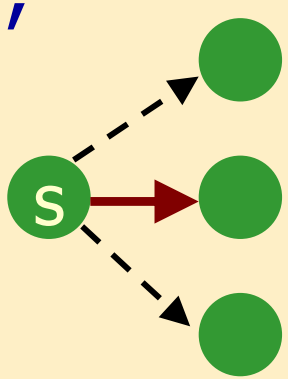
Számítási fa: Lehetséges elágazások



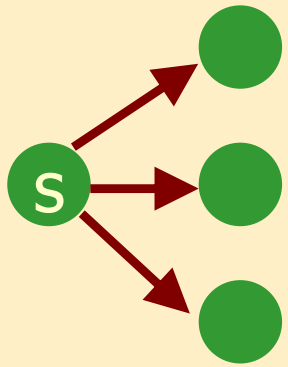
Elágazások vizsgálata

Egy-egy állapotban megvizsgálható,
hogy az útvonalakra vonatkozó követelmény
hány onnan kiinduló útvonal mentén teljesülhet:

- E p (Exists p): Létezik legalább egy útvonal, ahol a p követelmény teljesül
  - Egy lehetséges továbblépés mentén vizsgál
  - Egzisztenciális operátor



- A p (forAll p): Minden útvonalra fennáll, hogy a p követelmény teljesül
  - Minden lehetséges továbblépést magába foglal
  - Univerzális operátor



Elágazó idejű temporális logikák

- CTL*: Computational Tree Logic *
  - Útvonal kvantorok (E, A), és
  - útvonalakon az állapotokra vonatkozó temporális operátorok (F, G, X, U)

  tetszőleges kombinációja

- CTL: Computational Tree Logic
  - Állapotokra vonatkozó operátorokat mindig közvetlenül meg kell előznie útvonal kvantoroknak
  - Állapotokra vonatkozó operátorok nem kombinálhatók

CTL*: Computational Tree Logic *

CTL* operátorok (informális)

- Útvonalak kvantorai (állapotokon értelmezett):
  - A: „for All futures”,
    minden lehetséges útra az adott állapotból kiindulva
  - E: „Exists future”, „for some future”,
    legalább egy útra az adott állapotból kiindulva
- Állapotok operátorai (útvonalakon értelmezett):
  - F p: „Future”, valamikor az útvonal egy állapotán p igaz
  - G p: „Globally”, az útvonal minden állapotán p igaz
  - X p: „neXt”, a következő állapotban p igaz
  - p U q: „p Until q”, az útvonal egy állapotán igaz lesz q,
    és addig minden állapotban igaz p

CTL* kifejezések

$$A(p \Rightarrow F\ q)$$

Minden útvonalra igaz, hogy …

amennyiben p fennáll az útvonal elejétől, …

akkor ezt a jövőben olyan állapot fogja követni …

ahonnan indulva (az útvonal szuffixén) q fennáll.

Példa CTL* kifejezések

- $E(p \wedge G\ q)$

  Létezik olyan útvonal, hogy
  ezen p fennáll (az útvonal elejétől) és az útvonal minden szuffixén q is fennáll

- $E(XXX\ p \vee F\ q)$

  Létezik olyan útvonal, hogy
  - vagy ennek negyedik állapotán fennáll p,
  - vagy valamikor q fennáll az útvonalon

A CTL* formális kezelése

- Eddigiek: Csak informális bevezetés volt
- Az automatikus ellenőrzést is lehetővé tevő precíz megadáshoz szükséges:
  - Formális szintaxis szabályok:
    Mik az érvényes CTL* kifejezések?
  - Formális szemantika szabályok:
    Adott modellen mikor igaz egy CTL* kifejezés?

CTL* szintaxis

- Állapot-kifejezések: Állapotokon kiértékelhető
  - **S1**: Minden P atomi kijelentés egy állapot-kifejezés
  - **S2**: Ha p és q állapot-kifejezések, $\neg p$ és $p \wedge q$ is
  - **S3**: Ha p útvonal-kifejezés, akkor E p és A p állapot-kifejezések.

- Útvonal-kifejezések: Útvonalakon kiértékelhető
  - **P1**: Minden állapot-kifejezés útvonal-kifejezés
  - **P2**: Ha p és q útvonal-kifejezések, akkor $\neg p$ és $p \wedge q$ is
  - **P3**: Ha p és q útvonal-kifejezések, akkor X p és p U q is

Érvényes CTL* kifejezések:
A szabályok alapján generált állapot-kifejezések

„Kimaradt” operátorok

- true minden állapotra igaz
  false egy állapotra sem igaz
- $p \vee q$ jelentése $\neg((\neg p)\wedge(\neg q))$
  $p \Rightarrow q$ jelentése $(\neg p) \vee q$
  $p \equiv q$ jelentése $(p\Rightarrow q)\wedge(q\Rightarrow p)$
- F p jelentése true U p
  G p jelentése $\neg F(\neg p)$

CTL* szemantika: Jelölések

- $M=(S, R, L)$ Kripke-struktúra
- $\pi=(s_0, s_1, s_2,\ldots)$ az M egy útvonala, ahol s_0 a kezdőállapot és $\forall i\geq 0$: $(s_i, s_{i+1})\in R$
  - $\pi^i=(s_i, s_{i+1}, s_{i+2},\ldots)$ a π útvonal szuffixe i-től

- $M,\pi \models p$ jelentése (p útvonal-kifejezés): az M modellben a π útvonalon igaz p

- $M,s \models p$ jelentése (p állapot-kifejezés): az M modellben az s állapotban igaz p

CTL* szemantika: Állapot-kifejezések

- **S1**:

  $M,s \models P$ a.cs.a. $P \in L(s)$

- **S2**:

  $M,s \models \neg p$ a.cs.a. $M,s \models p$ nem igaz

  $M,s \models p \wedge q$ a.cs.a. $M,s \models p$ és $M,s \models q$

- **S3**:

  $M,s \models E\ p$ (ahol p útvonal-kifejezés)

  a.cs.a. létezik $\pi=(s_0, s_1, s_2,\ldots)$ útvonal M-ben

  $s= s_0$ mellett, hogy $M,\pi \models p$.

  $M,s \models A\ p$ (ahol p útvonal-kifejezés)

  a.cs.a. minden $\pi=(s_0, s_1, s_2,\ldots)$ útvonalra M-ben

  ahol $s= s_0$ fennáll igaz, hogy $M,\pi \models p$.

CTL* szemantika: Útvonal-kifejezések

- **P1**:

  $M,\pi \models p$ (p állapot-kifejezés) a.cs.a. $M, s_0 \models p$

- **P2**:

  $M,\pi \models \neg p$ a.cs.a. $M,\pi \models p$ nem igaz

  $M,\pi \models p \wedge q$ a.cs.a. $M,\pi \models p$ és $M,\pi \models q$

- **P3**:

  $M, \pi \models X\ p$ a.cs.a. $M,\pi^1 \models p$

  $M, \pi \models p\ U\ q$ a.cs.a

$$\exists j \geq 0 : (M, \pi^j \models q \text{ valamint } \forall 0 \leq k < j:\ M, \pi^k \models p)$$

Háttér: Az ellenőrzés komplexitása

- Worst-case időkomplexitás: Legalább $O(|S|^2 \times 2^{|p|})$
  - $|S|^2$ az átmenetek száma a modellben (Kripke-struktúrában) worst case esetben
  - $|p|$ az operátorok száma a temporális logikai kifejezésben
- Az exponenciális komplexitás riasztó
  - Bár a temporális logikai kifejezések tipikusan rövidek
- Célkitűzés: A CTL* egyszerűsítése
  - Gyakorlati problémák esetén használható maradjon
  - Az ellenőrzés worst-case időkomplexitása csökkenjen

CTL: Computational Tree Logic

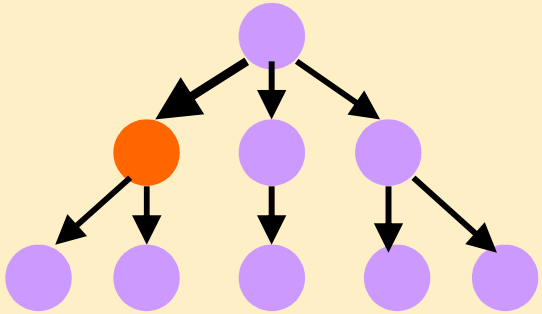
CTL operátorok (informális bevezető)

Állapotokon értelmezhető összetett operátorok:

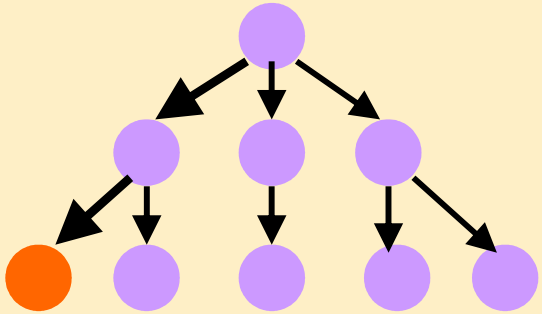
- EX p: létezik útvonal, aminek következő állapotán p
- EF p: létezik útvonal, aminek egy állapotán p
- EG p: létezik útvonal, aminek minden állapotán p
- E(p U q): létezik útvonal, amin p amíg q

- AX p: minden útvonal következő állapotán p
- AF p: minden útvonal egy-egy elérhető_állapotán p
- AG p: minden útvonal minden állapotán p
- A(p U q): minden útvonalon p amíg q

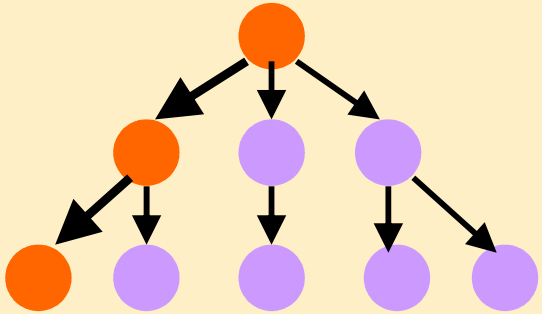
CTL operátorok (példák)



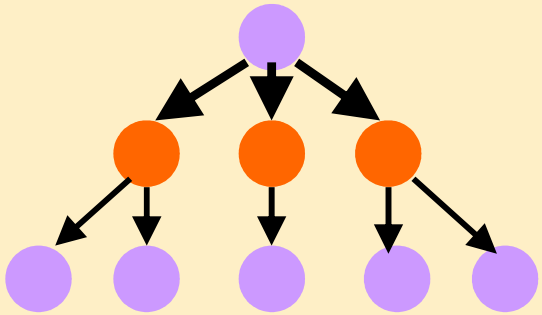
EX p



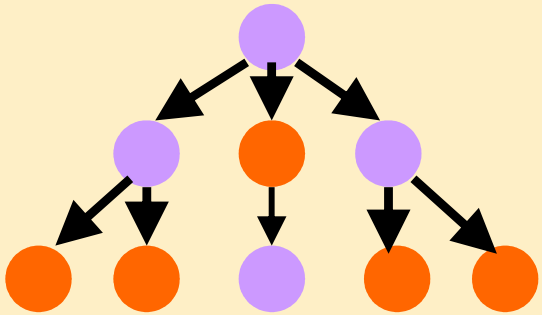
EF p



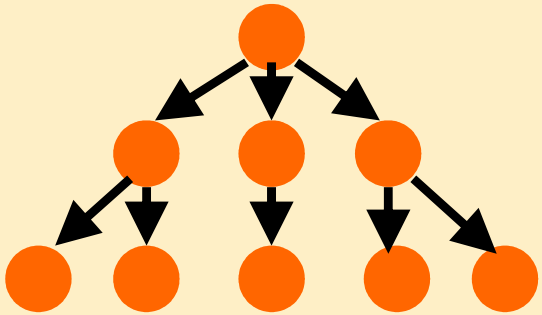
EG p



AX p



AF p

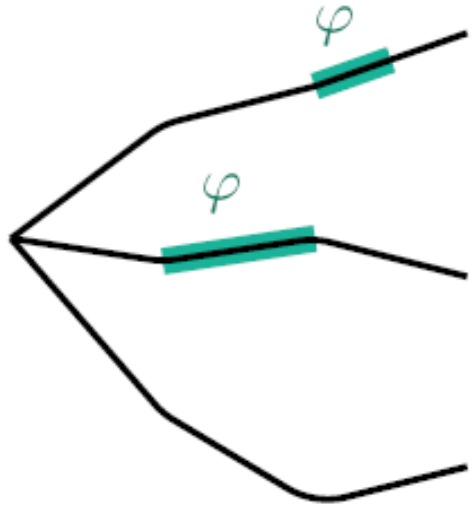


AG p

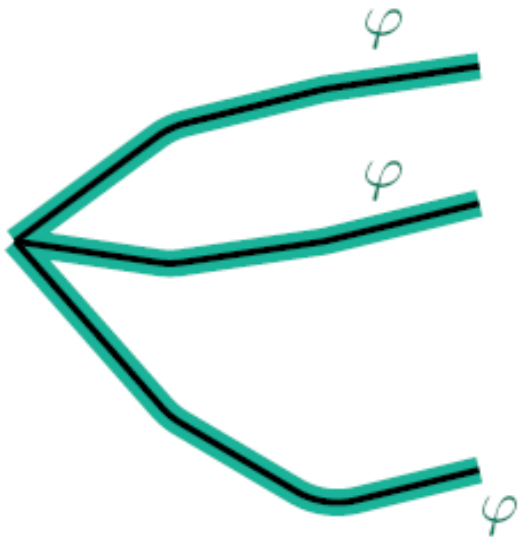
CTL operátorok (példák)

EG φ

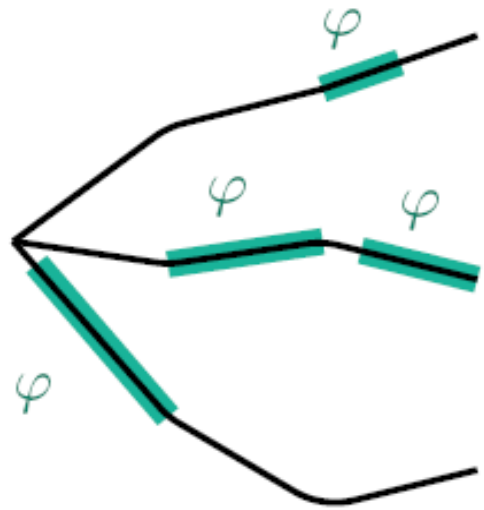
EF φ



AG φ



AF φ



CTL kifejezések (példák)

- AG EF p

  Bárhonnan indulva olyan állapotba vihető a rendszer, ahol p igaz

  - Pl. AG EF Reset

- AG AF p

  Bárhonnan indulva mindenképpen elérhetünk olyan állapotba, ahol p igaz

  - Pl. AG AF Terminated

- AG (p $\Rightarrow$ AF q)

  Bárhonnan indulva teljesül, hogy ha ott p igaz, akkor valamikor mindenképpen elérhető olyan állapot, ahol q igaz.

  - Pl. AG (Request $\Rightarrow$ AF Reply)

CTL kifejezések (példák)

- EF AG p

  Lehetséges, hogy a rendszer olyan állapotba kerül, hogy utána p minden állapotban igaz lesz

- AF AG p

  Bármely úton eljuthatunk olyan állapotba, hogy utána p mindig igaz lesz

- AG (p $\Rightarrow$ A (p U q))

  Bármelyik elérhető állapotban ha p igaz, akkor minden úton p fennáll q eléréséig

  - „p fennáll q eléréséig” pontosabban: elérünk egy olyan állapotba, ahol q igaz, és addig minden állapotban p igaz

CTL formális szintaxis I.

Állapot-kifejezések:

- CTL* esetén volt:
  - **S1**: Minden P atomi kijelentés egy állapot-kifejezés
  - **S2**: Ha p és q állapot-kifejezések, $\neg p$ és $p \wedge q$ is
  - **S3**: Ha p útvonal-kifejezés, akkor E p és A p állapot-kifejezések.

- CTL esetén ezek (**S1**, **S2**, **S3**) változatlanok!

CTL formális szintaxis II.

Útvonal-kifejezések:

- CTL* esetén volt:
  - **P1**: Minden állapot-kifejezés útvonal-kifejezés
  - **P2**: Ha p és q útvonal-kifejezések, akkor $\neg p$ és $p \wedge q$ is
  - **P3**: Ha p és q útvonal-kifejezések, akkor X p és p U q is

- CTL esetén ezek helyett egy szabály lesz:
  - **P0**:
    - Ha p és q <u>állapot-kifejezések</u>,
      akkor X p és p U q útvonal-kifejezések.

CTL formális szintaxis III.

Következmények:

- Útvonal-kifejezések nem kombinálhatók:
  - X és U csak állapot-kifejezéseken alkalmazhatók
  - Boole operátorok csak állapot-kifejezéseken alkalmazhatók
- Útvonal-kifejezéseket csak az **S3** szabály használja
- Az **S3** szabály miatt X p és p U q útvonal-kifejezések elé rögtön egy útvonal kvantor kerül
  (mást nem tehetünk útvonal-kifejezésekkel):
  ezért „összenőnek” az operátorok
  - EX, E(. U .),
  - AX, A(. U .)

„Kimaradt” operátorok

- EF p jelentése E(true U p)
- AF p jelentése A(true U p)
- EG p jelentése $\neg$AF ($\neg$p)
- AG p jelentése $\neg$EF ($\neg$p)

CTL* de nem CTL

- E(XXX Piros)
  - A többszörös X operátor használat miatt egymásba ágyazott útvonal-kifejezések vannak,
  - azaz a két külső X operátort útvonal-kifejezésre alkalmaztuk
- E(X Piros $\vee$ F Sárga)
  - Boole operátor van útvonal-kifejezések között
- A(X G (Piros $\wedge$ Sárga))
  - Egymásba ágyazott útvonal-kifejezések vannak

CTL formális szemantika

- Állapot-kifejezések:
  - **S1**, **S2**, **S3** szabályok (lásd CTL*) változatlanok
- Útvonal-kifejezések:
  - **P1**, **P2**, **P3** helyébe egy új **P0** szabály lép:

P0:

- $M,\pi \models X\ p$ ahol p állapot-kifejezés a.cs.a. $M,s_1 \models p$
- $M,\pi \models p\ U\ q$ ahol p,q állapot-kifejezés a.cs.a.

$$\exists j \geq 0 : (M, s_j \models q \text{ valamint } \forall 0 \leq k < j:\ M,s_k \models p)$$

Itt állapot-kifejezések vannak a szintaxis szabály szerint!

Egy példa

- Két processzből álló rendszer: P1 és P2
- Processz állapotok a követelmények szempontjából:
  - Kritikus szakaszban van: C1, C2
  - Nem-kritikus szakaszban van: N1, N2
  - Kritikus szakaszba belépésre kész: W1, W2
- Atomi kijelentések:
  AP = {C1, C2, N1, N2, W1, W2}

Mintapélda (folytatás)

- Egyszerre csak egy processz lehet a kritikus szakaszban:
  $AG\ (\neg(C1 \wedge C2))$
- Ha egy processz be akar lépni a kritikus szakaszba, akkor előbb-utóbb mindig beléphet:
  $AG\ (W1 \Rightarrow AF(C1))$
  $AG\ (W2 \Rightarrow AF(C2))$
- A processzek felváltva kerülnek a kritikus szakaszba; egyikük kilép majd a másik lép be:
  $AG(C1 \Rightarrow A(C1\ U\ (\neg C1 \wedge A((\neg C1)\ U\ C2))))$
  $AG(C2 \Rightarrow A(C2\ U\ (\neg C2 \wedge A((\neg C2)\ U\ C1))))$

Mintapélda (folytatás)

- Egyszerre csak egy processz lehet a kritikus szakaszban:
  AG (¬(C1 ∧ C2
- Ha egy process ni a kritikus szakaszba, akkor előbb-utóbb mind belé
  (C1))
  (C2))
- A process k felváltva kerülnek a kri kus szakaszba; egyikük kilép majd a másik lép be:
  AG(C1 ⇒ A(C1 U (¬C1 ∧ A((¬C1) U C2))))
  AG(C2 ⇒ A(C2 U (¬C2 ∧ A((¬C2) U C1))))

P1 nincs a kritikus szakaszban

P2 lép a kritikus szakaszba

P1 van a kritikus szakaszban

Háttér: Az ellenőrzés komplexitása

- Worst case időkomplexitás: $O(|S|^2 \times |p|)$
  - $|S|^2$ az átmenetek száma a modellben (Kripke-struktúra) worst case esetben
  - $|p|$ az operátorok száma a temporális logikai kifejezésben
- CTL*-nál kedvezőbb a CTL esetén:
  - Itt nincs $2^{|p|}$ tag
  - Sok gyakorlati követelmény esetén jól használható
    - Biztonsági követelmények: AG
    - Élőségi követelmények: EF, AF

Kifejezőképesség

- Egy temporális logika kifejezőképessége nagyobb egy másikénál, ha képes olyan tulajdonság megfogalmazására, amire a másik nem.
- Eddigi tapasztalatok:
  - Lineáris idejű logika nem tudja figyelembe venni a lehetséges elágazásokat
    (implicit „minden útra” jellegű vizsgálat lehetséges)
  - CTL kötöttebb, mint a CTL*,
    ezért kevesebb tulajdonság megfogalmazására képes
  - CTL* magába foglalja a lehetséges PLTL kifejezéseket

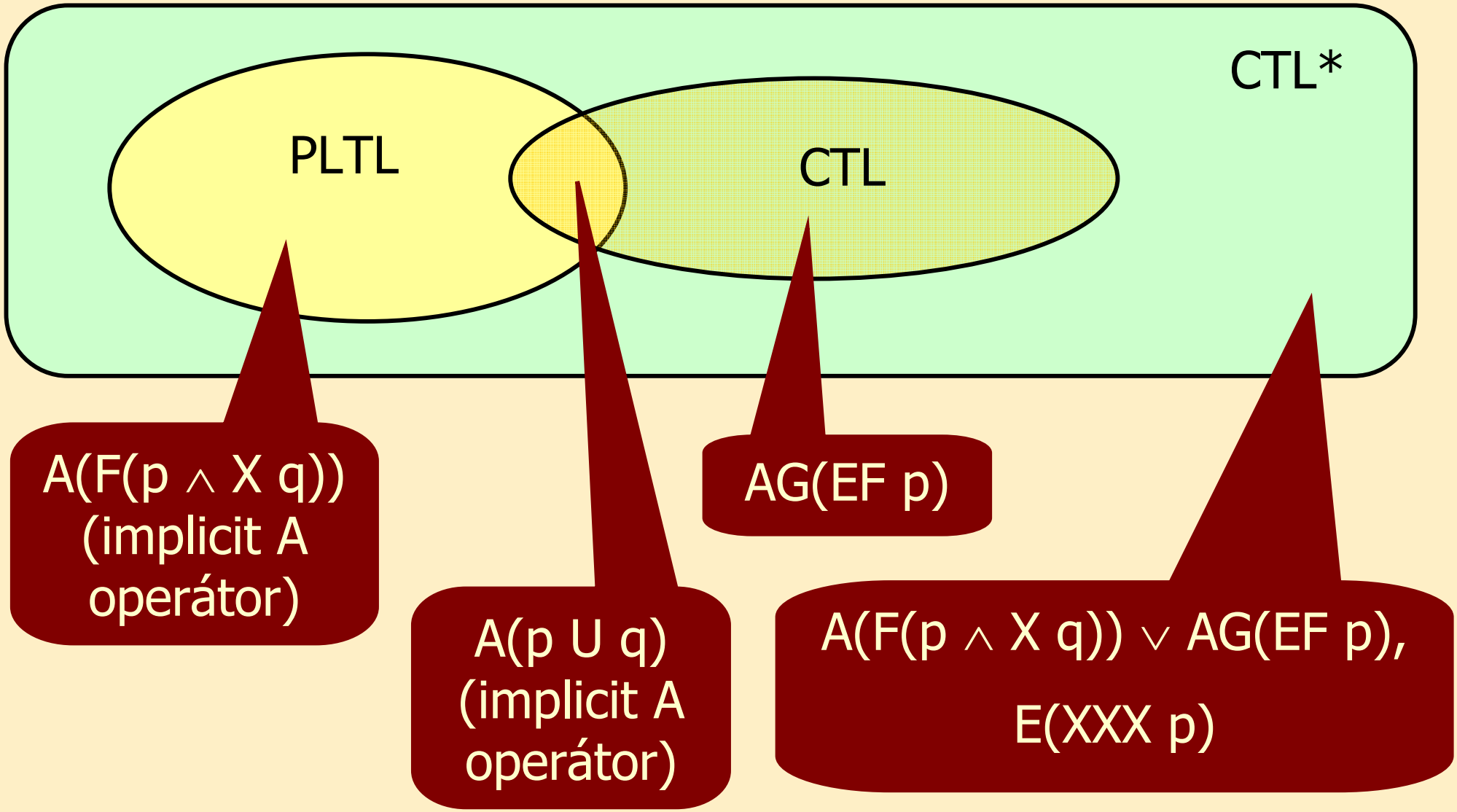
Kifejezőképesség - formálisan

- TL1 és TL2 logikák kifejezőképessége azonos, ha minden M modellre és annak minden s állapotára:

$$\forall p \in TL1:$$
$$\exists q \in TL2:\ (M, s \models p \Leftrightarrow M, s \models q),$$
$$\forall q \in TL2:$$
$$\exists p \in TL1:\ (M, s \models p \Leftrightarrow M, s \models q)$$

- Ha csak az első tag igaz,
akkor TL1 kevésbé kifejező, mint TL2

LTL, CTL, CTL* kifejezőképessége



Kitekintés

Sztochasztikus logikák:

- Megbízhatósági illetve idő követelményekhez használható
  - Pl.: Ha HIBA az aktuális állapot, akkor ez kisebb mint 30% valószínűséggel áll fenn 2 időegység múlva is
- CTL kiterjesztése:
  - Folytonos idejű Markov-láncokon értelmezett (nem Kripke-struktúra)
  - Valószínűségi kritérumok állapotok eléréséhez (állandósult állapot), útvonalak bejárásához
  - Idő kritériumok (időintervallumok) X és U operátorokhoz

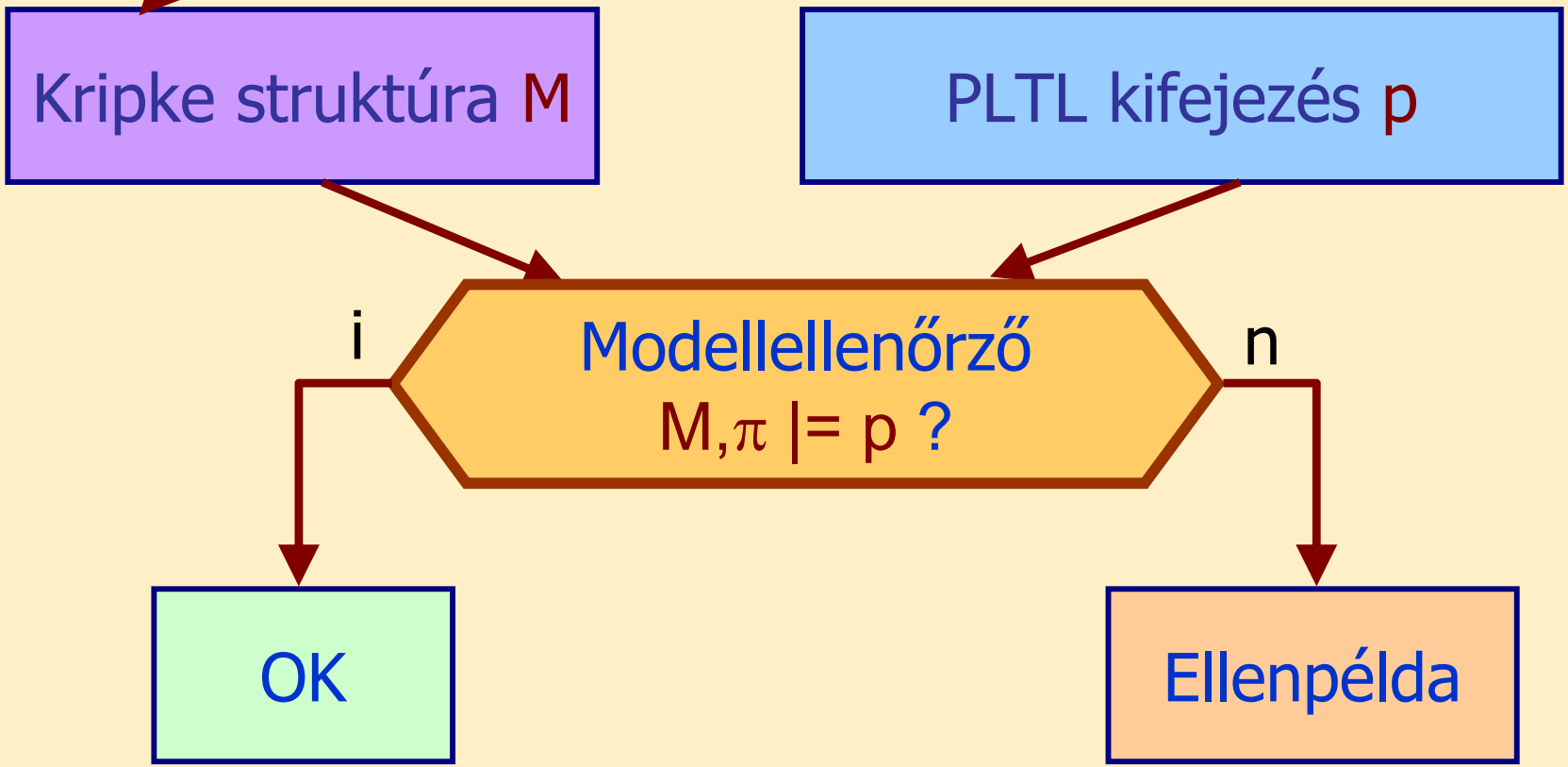
Valósidejű logikák:

- Valósidejű rendszerek követelményeinek megadásához
  - Óraváltozókra hivatkozhat a logika
  - Időintervallumok kezelése kidolgozott

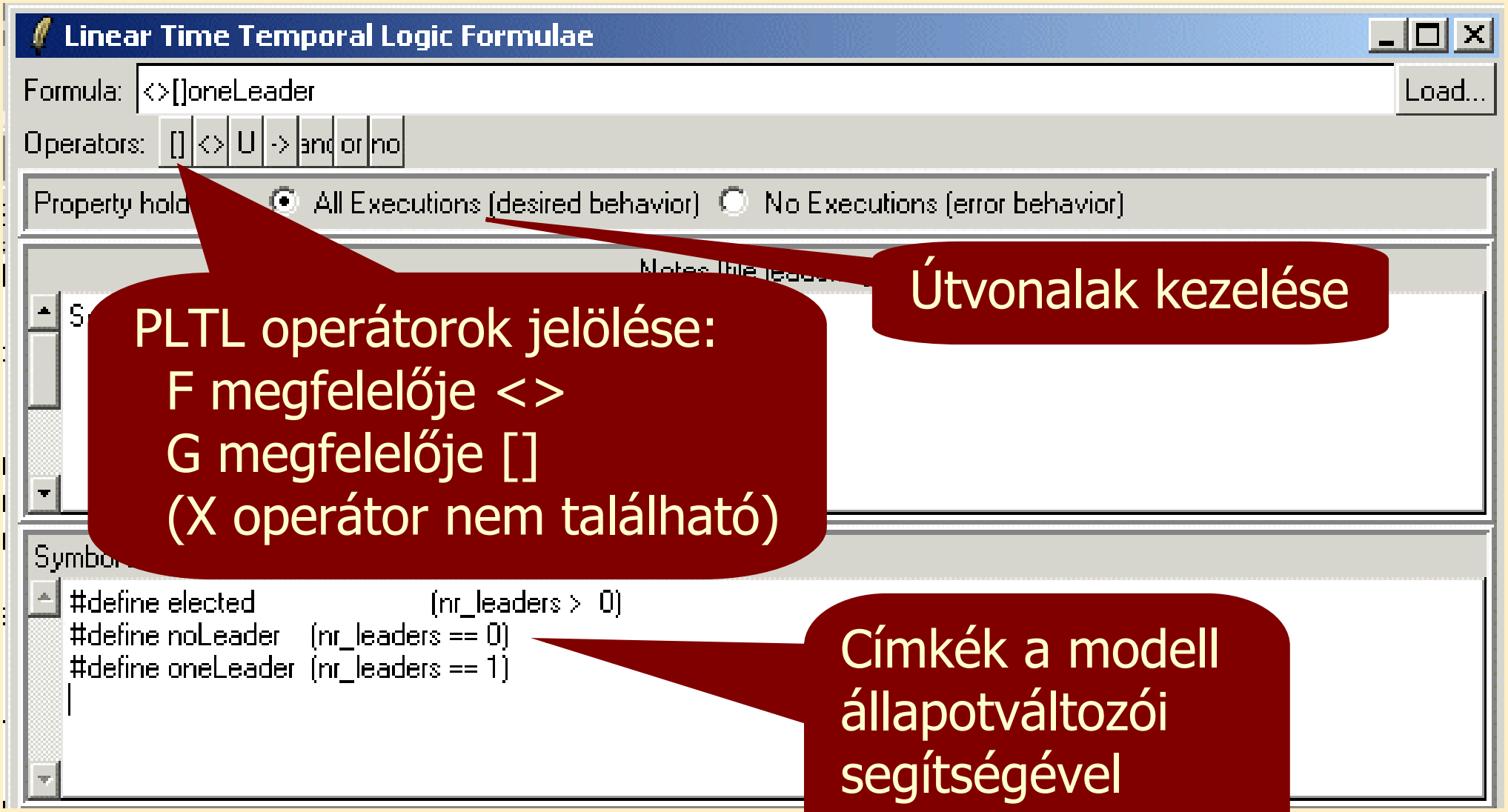
A modellellenőrzési feladat

PLTL modellellenőrzés

Ha nincs útvonal megadva, akkor a kezdőállapotból induló minden útra ellenőriz!



A SPIN modellellenőrző



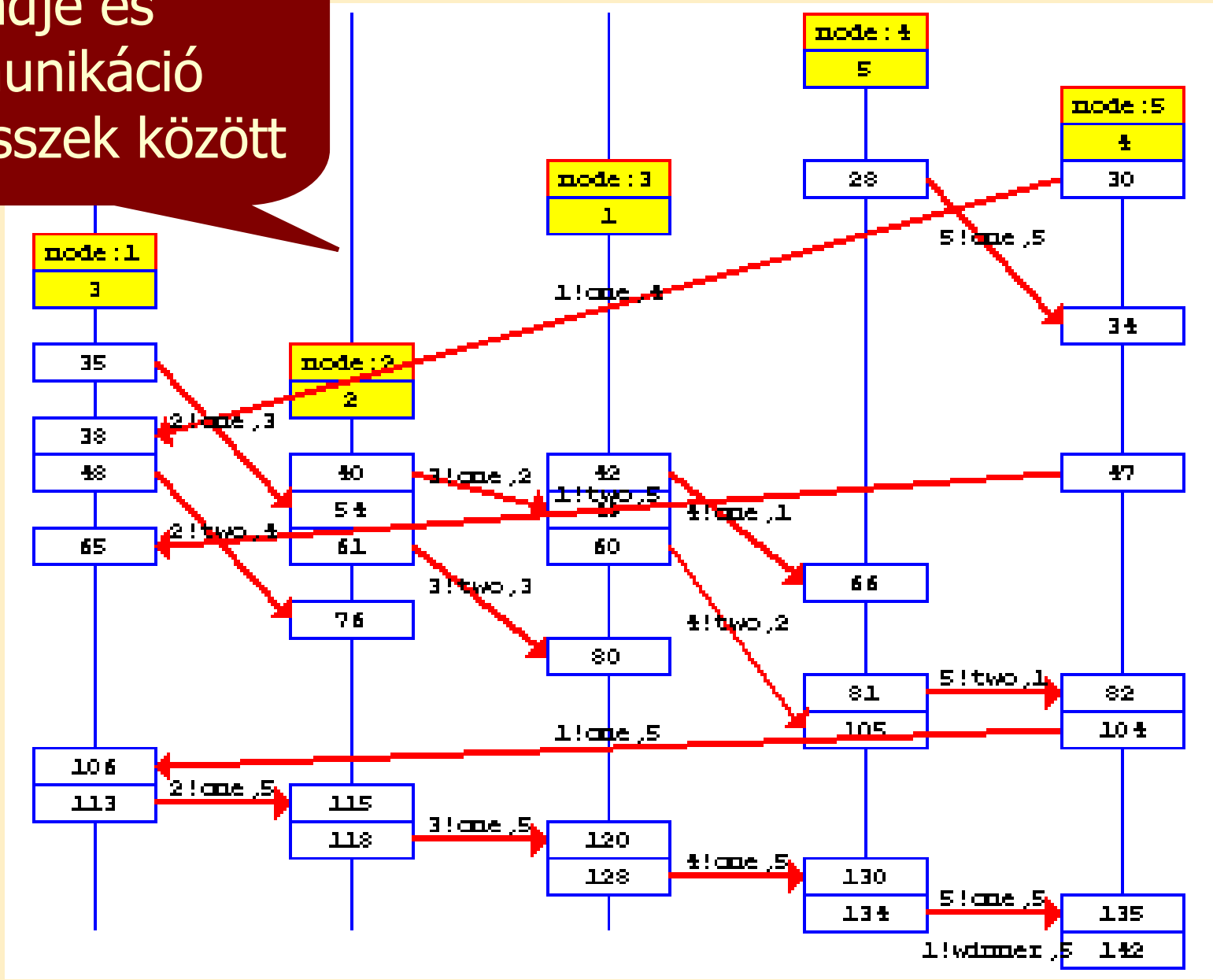
Útvonalak kezelése

PLTL operátorok jelölése:
F megfelelője <>
G megfelelője []
(X operátor nem található)

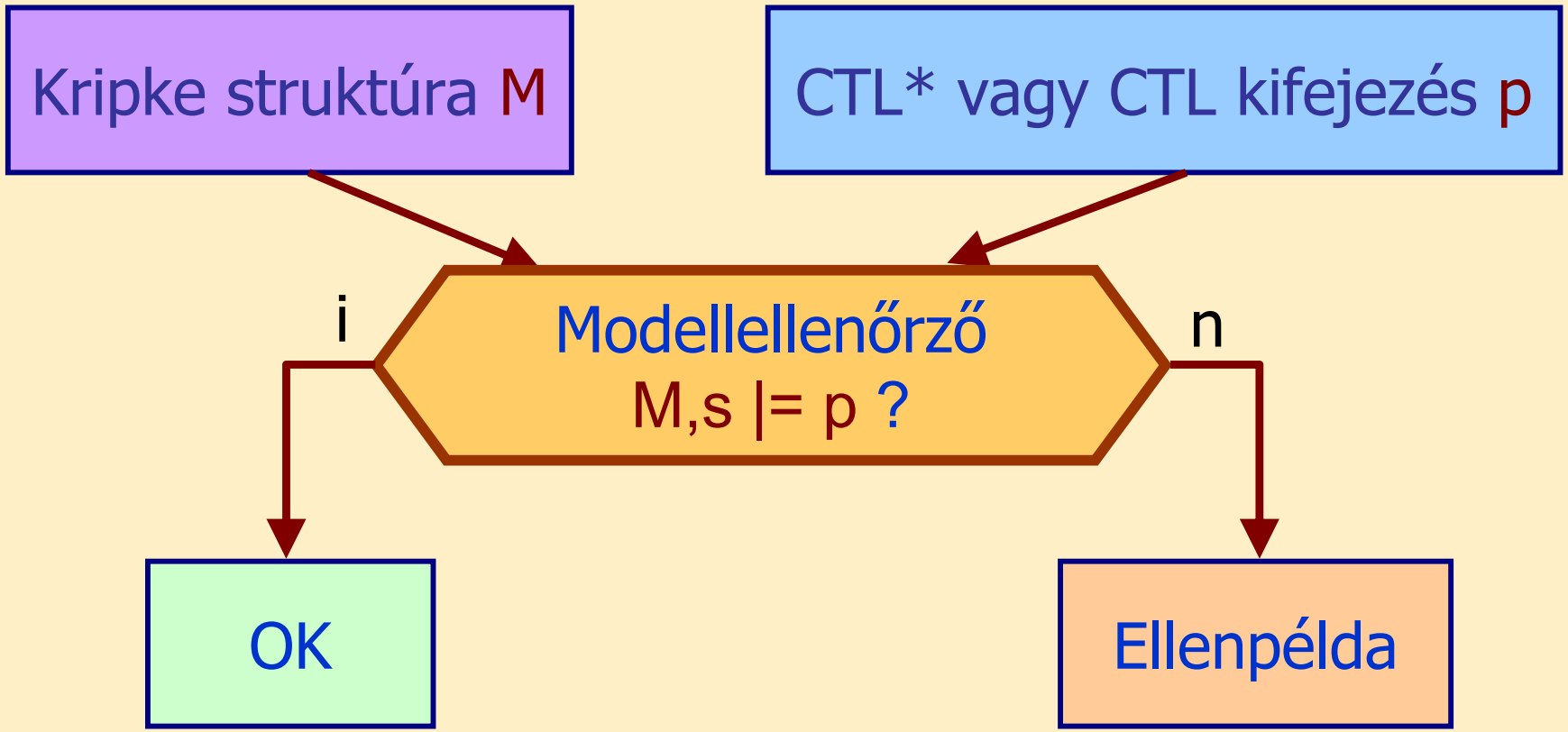
Címkék a modell állapotváltozói segítségével definiálhatók

SPIN ellenpélda

Itt: Állapotok sorrendje és kommunikáció processzek között



CTL* vagy CTL modellellenőrzés



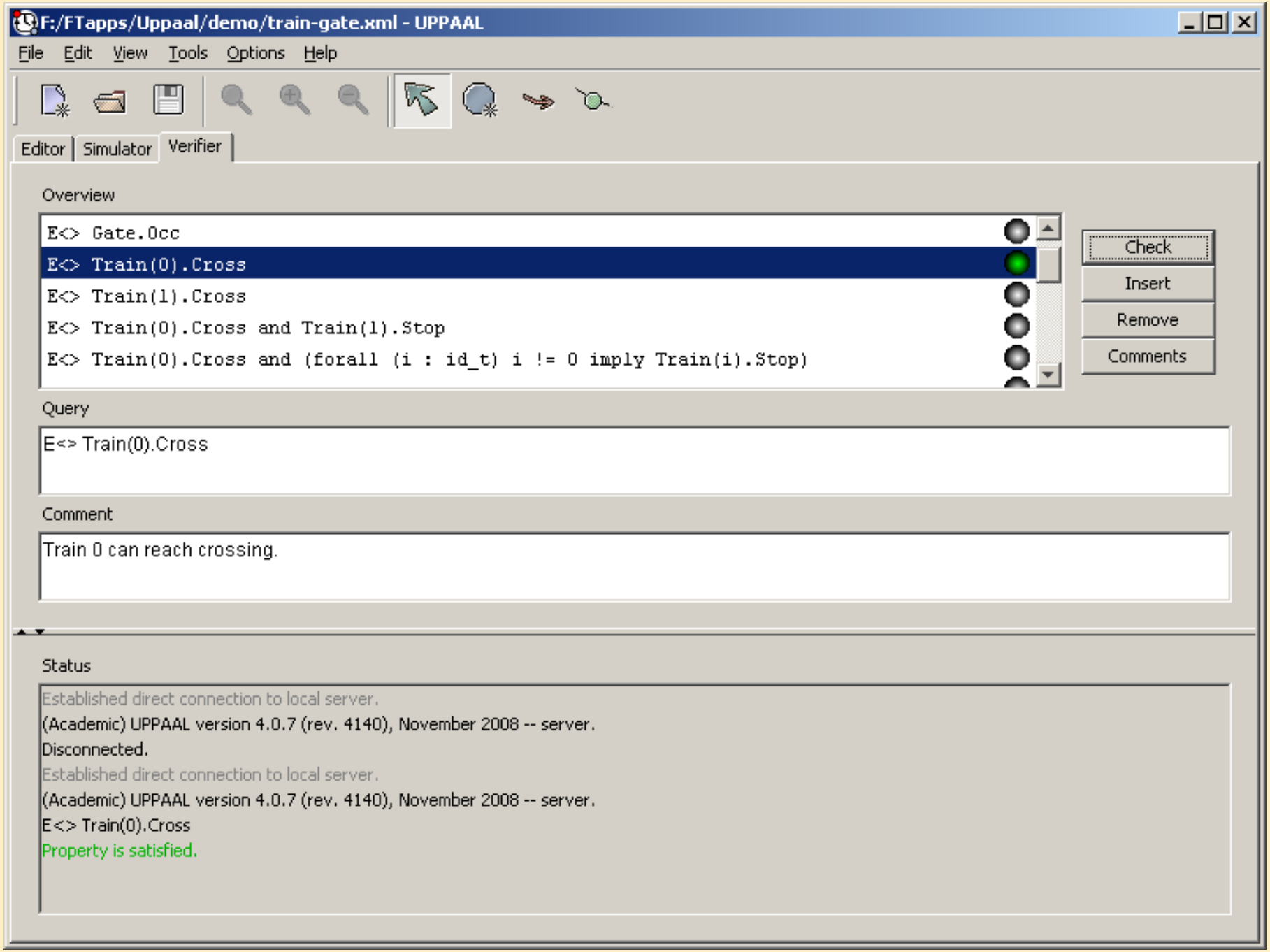
Az UPPAAL modellellenőrző lehetőségei

- Atomi kijelentések:
  - Változók értéke hivatkozható: pl. a!=1
    - Egész aritmetika és bitenkénti műveletek használhatók
  - Állapot hivatkozható: pl. Train(0).cross
    - Paraméterezett processzekre: forall, exists operátorok
  - Holtpont: deadlock kijelentéssel megadható (nincs akció)
- Boole logikai operátorok:
  - and, or, imply, not, ? : (ez utóbbi az if-then-else)
- Temporális operátorok: Korlátozott CTL
  - Jelölés: [] szerepel G helyett, <> szerepel F helyett
    - Így lesz: A[], A<>, E[], E<>
    - E[] esetén véges útvonalon is értelmezett (végállapotig)
  - Egymásba nem ágyazhatók temporális operátorok
    - Plusz lehetőség: p-->q rövidítés jelentése A[] (p imply A<> q)

Követelmények ellenőrzése az UPPAAL-ban

- Követelmények halmaza szerkeszthető
- Modell ellenőrzés egyenként is indítható
- Ellenpélda generálható:
  - Legrövidebb, leggyorsabb, vagy akármi
  - Betölthető a szimulátorba (végigjátszható)
- Keresés az állapottérben:
  - Mélységi
  - Szélességi
- Állapottárolás különféle opciókkal:
  - Redukció
  - Közelítő állapottér tárolás (alul- illetve felülbecslés)
  - Hash tábla mérete megadható

Az UPPAAL modellellenőrző ablaka



Ellenpélda az UPPAAL szimulátorban

F:/FTapps/Uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor | Simulator | Verifier

Drag out

Enabled Transitions

- appr[2]: Train(2) --> Gate
- appr[3]: Train(3) --> Gate
- appr[4]: Train(4) --> Gate
- appr[5]: Train(5) --> Gate
- leave[0]: Train(0) --> Gate

Next | Reset

Simulation Trace

- (Safe, Safe, Safe, Safe, Safe, Safe, Free)
- appr[0]: Train(0) --> Gate
- (Appr, Safe, Safe, Safe, Safe, Safe, Occ)
- Train(0)
- (Cross, Safe, Safe, Safe, Safe, Safe, Occ)
- appr[1]: Train(1) --> Gate
- (Cross, Appr, Safe, Safe, Safe, Safe, -)
- stop[tail()]: Gate --> Train(1)
- (Cross, Stop, Safe, Safe, Safe, Safe, Occ)

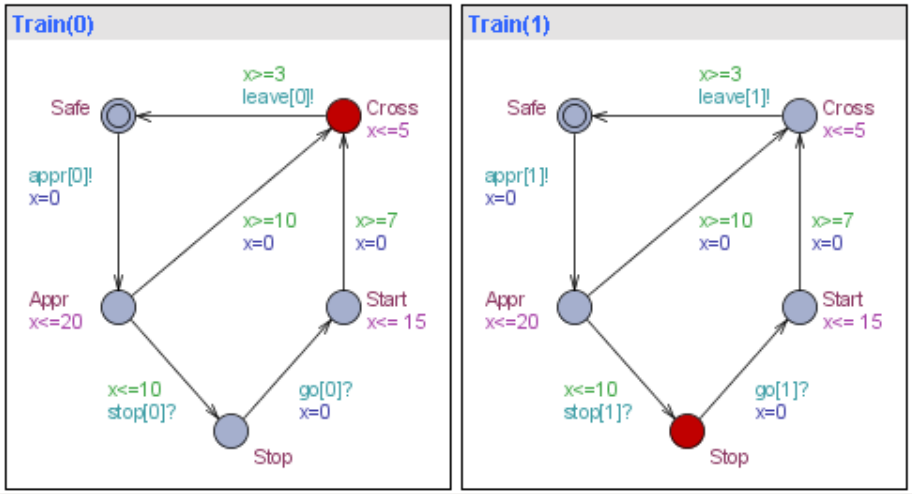
Trace File:

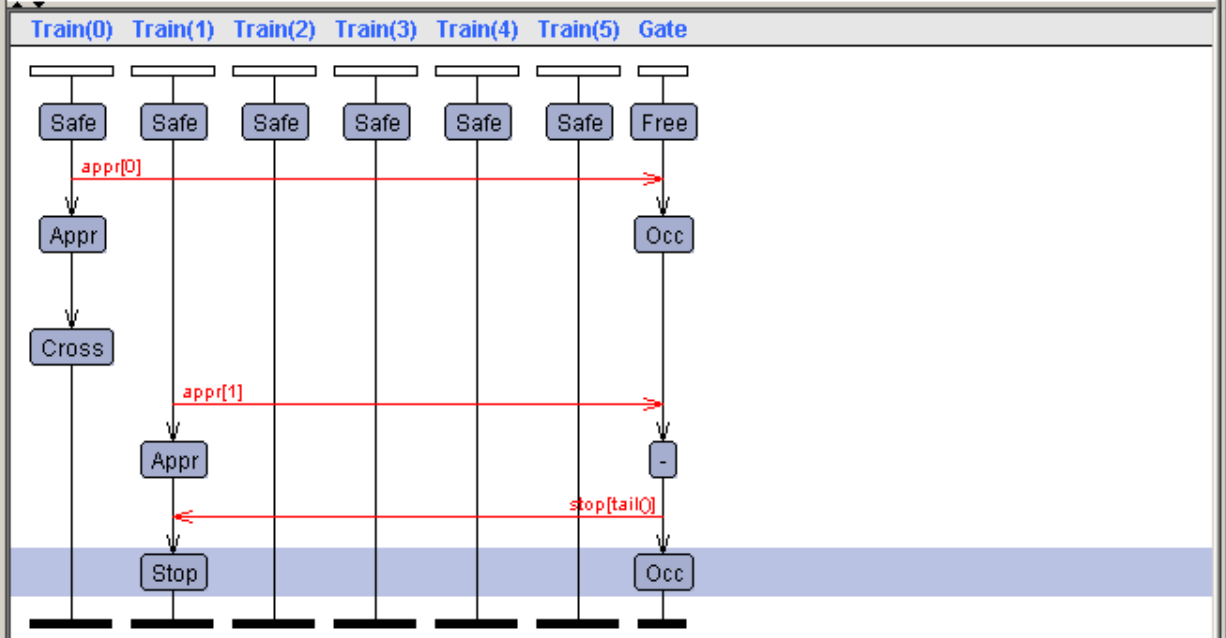
Prev | Next | Replay | Open | Save | Auto

Slow — Fast

Drag out

- Gate.list[0] = 0
- Gate.list[1] = 1
- Gate.list[2] = 0
- Gate.list[3] = 0
- Gate.list[4] = 0
- Gate.list[5] = 0
- Gate.list[6] = 0
- Gate.len = 2
- Train(0).x in [0,5]
- Train(1).x in [0,5]
- Train(2).x >= 10
- Train(3).x >= 10
- Train(4).x >= 10
- Train(5).x >= 10
- Train(0).x - Train(2).x <= -10
- Train(1).x - Train(0).x in [-5,0]
- Train(2).x = Train(3).x
- Train(3).x = Train(4).x
- Train(4).x = Train(5).x
- Train(5).x = Train(2).x





A mintapélda befejezése

Mintapélda: Kölcsönös kizárás

- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

P1

```
while (true) {
    blocked1 = true;
    while (turn!=1) {
        while (blocked0==true) {
            skip;
        }
        turn=1;
    }
    // Critical section
    blocked1 = false;
    // Do other things
}
```

Helyes-e ez az algoritmus?

A modell UPPAAL-ban

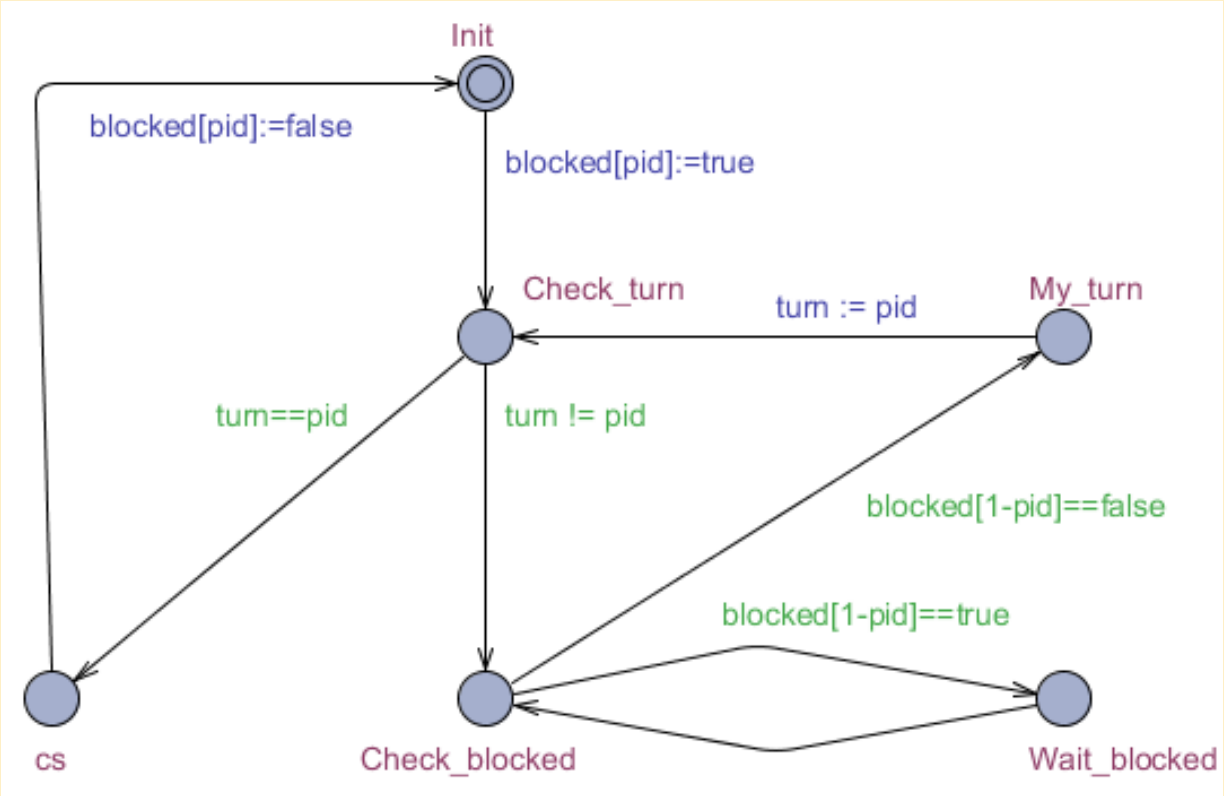
Deklarációk:

```
int[0,1] blocked[2];
int[0,1] turn;
P0 = P(0);
P1 = P(1);
system P0,P1;
```

Kihasznált modellezési lehetőségek:

- Közös változók rendszerszintű deklarálása
- Azonos viselkedésű résztvevők azonos automata template alapján
- Példányosítás paraméterezéssel
- Korlátozott értékkészletű változók
- Változó tömbök (résztvevőkhöz)

A P automata:



P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

UPPAAL: A követelmények formalizálása

- Kölcsönös kizárás:
  - Egyszerre csak az egyik résztvevő lehet a kritikus szakaszban: A[] not (P0.cs and P1.cs)
- Holtpontmentesség:
  - Nem alakul ki kölcsönös várakozás (leállás): A[] not deadlock
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán beléphet a kritikus szakaszba: E<>(P0.cs)
  - P1 egyáltalán beléphet a kritikus szakaszba: E<>(P1.cs)
- Nincs kiéheztetés:
  - P0 mindenképpen be fog lépni a kritikus szakaszba: A<>(P0.cs)
  - P1 mindenképpen be fog lépni a kritikus szakaszba: A<>(P1.cs)

UPPAAL: A modellellenőrzés eredménye

- Nincs holtpont
- Az élő jellegű követelmények teljesülnek
- Kiéheztetés elkerülése időzítések megadása nélkül nem vizsgálható
  - Triviális ellenpélda: A kiinduló állapotban marad
    - Committed állapot vagy állapot invariáns szükséges
    - Ez a valósidejű ellenőrzés „specialitása”
  - Ezután is lehet kiéheztetés?
- A kölcsönös kizárás nem teljesül!
  - Ellenpélda: Átlapolódás a két résztvevő között (végigjátszható a szimulátorban)

Az algoritmus javítása

Peterson algoritmusa

- P0 résztvevőre (P1 értelemszerű):

Hyman:

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

Peterson:

```
while (true) {
    blocked0 = true;
    turn=1;
    while (blocked1==true &&
        turn!=0) {
            skip;
    }

    // Critical section
    blocked0 = false;
    // Do other things
}
```

Összefoglalás

- Lineáris idejű temporális logikák:
  - PLTL
- Elágazó idejű temporális logikák:
  - CTL*
  - CTL (kötöttebb, de egyszerűbben ellenőrizhető)
- Formális szintaxis és szemantika
- Modellellenőrzési feladat
  - Megoldás algoritmusa: Következő előadás!