2.előadás

Források:

-Molnár Ágnes: Formális módszerek az informatikában (1) , NetAkadámia Tudástár

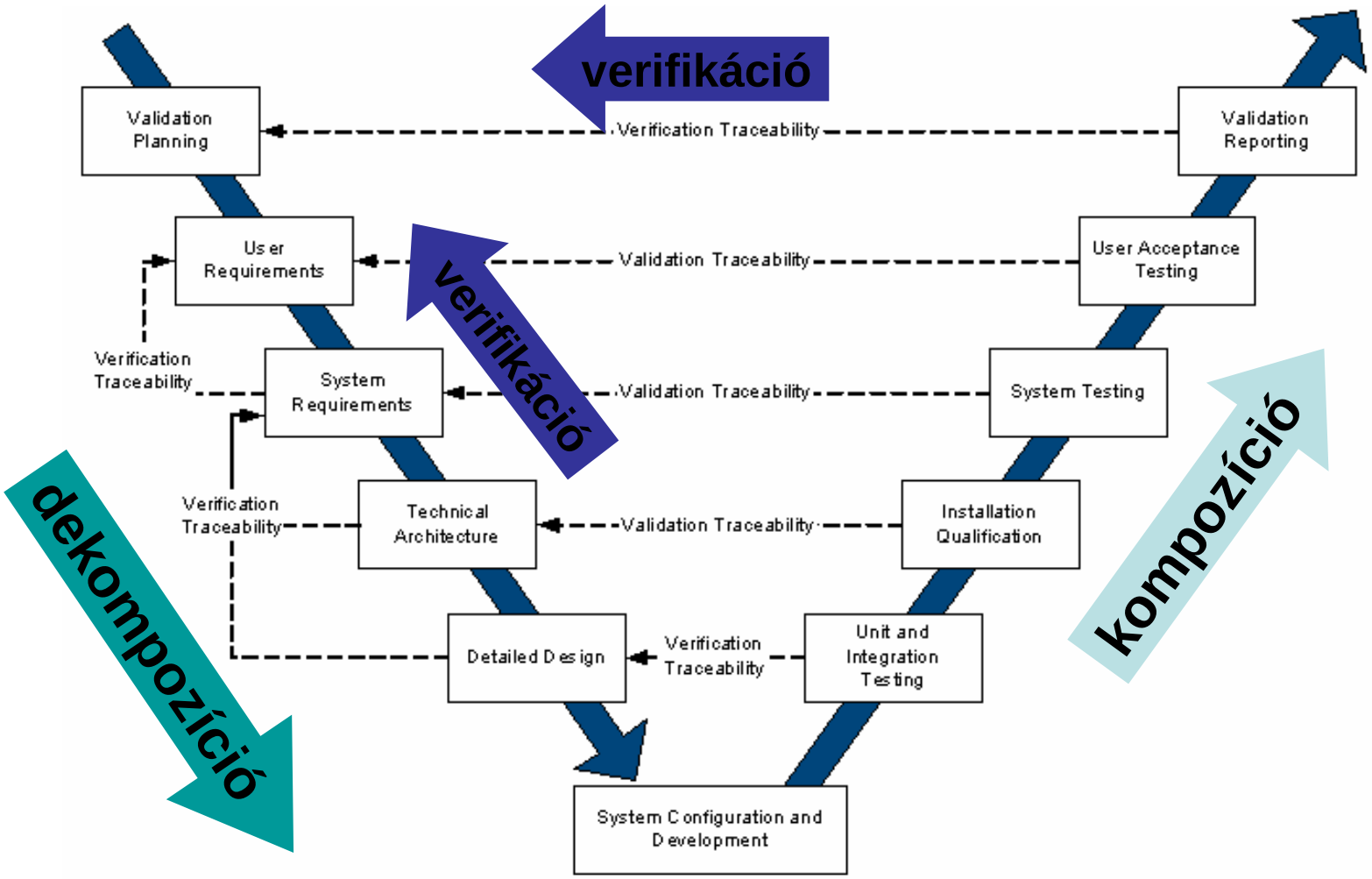
-dr. Pataricza András, dr. Bartha Tamás: Petri hálók: alapfogalmak, formális definíció

Validáció és verifikáció

- Tekintve, hogy az informatikai rendszerek jellegzetesen egyediek, a minőségbiztosításnak nem a termék reprodukciójára, hanem magára a konstrukcióra kell koncentrálnia.
- A tervezés során a végcél egy olyan rendszer felépítése, amely bizonyítottan **helyes** szolgáltatásokat nyújt a végfelhasználó felé.
- E cél érdekében *egyrészt alkalmazhatunk matematikai módszereket*, amelyek (az adott modell érvényességén belül) 100% valószínűséggel bizonyítják a struktúra helyességét.
- *Másrészt tervezett teszteléssel* is megpróbálhatjuk megbecsülni épülő vagy már kész rendszerünk jóságát.
- Sajnos azonban a legalaposabb tesztelés sem képes a teljes problématér átvizsgálására, ezért sohasem adhat 100%-os bizonyosságot rendszerünk helyes működéséről.

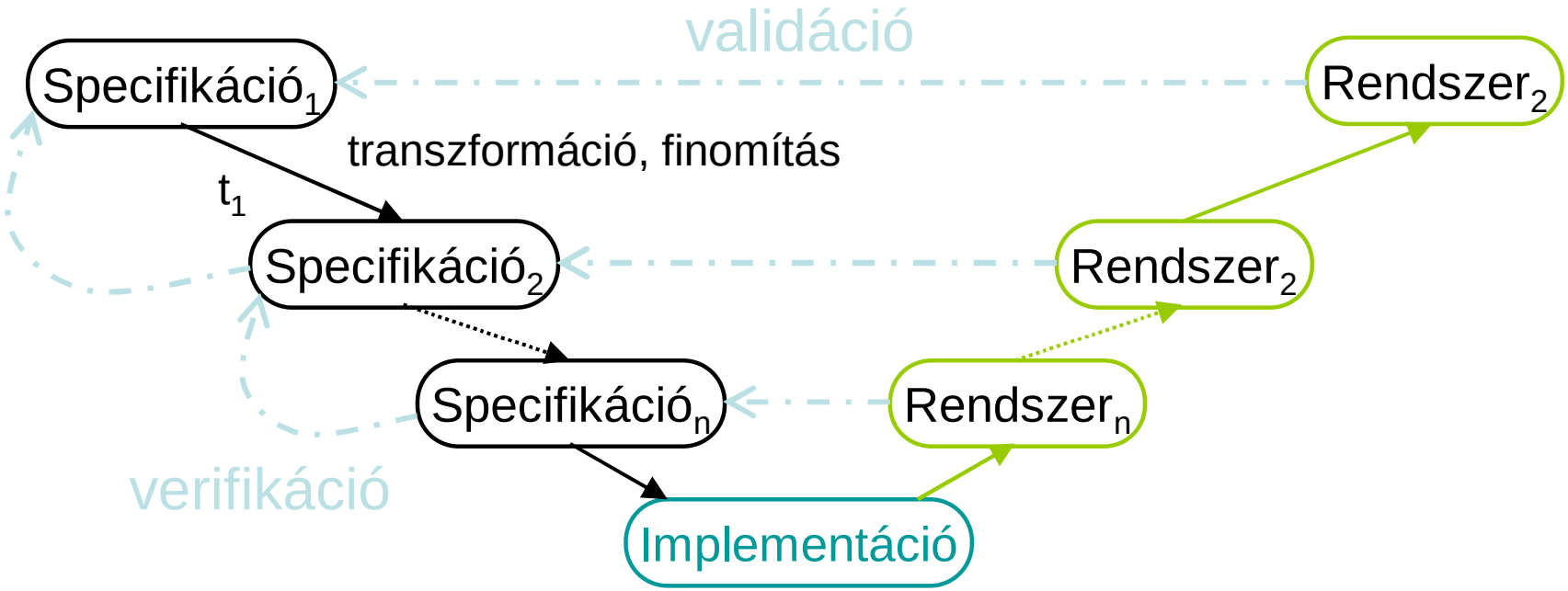
- A vizsgálat során két kérdéskört kell alaposan végigjárnunk:
- a megrendelő igényeinek megfelelő, *jó* rendszert építünk-e (*validáció*),
- illetve a fejlesztés során kapott különböző mélységű modellek ekvivalensek-e, azaz *jól* építjük-e a rendszert (*verifikáció*).

V modell


verifikáció
Validation Planning
Verification Traceability
Validation Reporting
User Requirements
Validation Traceability
User Acceptance Testing
verifikáció
Verification Traceability
System Requirements
Validation Traceability
System Testing
dekompozíció
Verification Traceability
Technical Architecture
Validation Traceability
Installation Qualification
kompozíció
Detailed Design
Verification Traceability
Unit and Integration Testing
System Configuration and Development

Verifikáció és validáció szerepe

- **Verifikáció**:
  - jól építjük-e a rendszert?
- **Validáció**:
  - jó rendszert építünk-e?



Tegyük fel, hogy biztosak lehetünk abban, hogy a kiinduló specifikációnk teljes (az egész problémateret lefedi), és az összes megrendelői kívánalmat teljesíti.

- Ebben az esetben természetesen elég a tisztán **verifikációs** megközelítés, azaz a specifikáció és az implementációs modellek ekvivalenciájának bizonyítása.

A valós életben azonban gyakran előfordul, hogy a fenti feltételek nem teljesülnek, a rendszer biztonságkritikus, vagy a felhasznált eszközkészlet által nem garantált tulajdonságok ellenőrzése elkerülhetetlen (pl. holtpontmentesség), a **validációt** sem kerülhetjük el.

A specifikáció teljességének ellenőrzése már önmagában is kreativitást igénylő feladat, hiszen bonyolultabb rendszerek reakcióit nemcsak az adott esetben funkcionálisan értelmes esetekre kell specifikálni, hanem ***minden*** **elképzelhető (és el sem képzelhető)** bemeneti szekvenciára is.

Modellalkotás és a modell ellenőrzése

- A fejlesztés első lépéseként a felhasználó igényeinek megfelelő rendszer egy véges modelljét kell megalkotnunk.
- Az informatikai *modell* a számítógépben tárolható adatszerkezetek, adatstruktúrák és az ezeket kezelő, átalakító, ezekből információkat lekérdező algoritmusok összessége, rendszere.
- A *modellalkotás* egy absztrakciós folyamat, amelynek során egyrészt elhagyjuk a feladat szempontjából lényegtelen jellemzőket, másrészt pontosítjuk és formalizáljuk a lényegeseket, állandóan szem előtt tartva a feladatot majd megoldó eszköz (jelen esetben egy számítógépes hardver-szoftver környezet) tulajdonságait és képességeit.

- Egy bonyolultabb probléma megoldása közben tapasztalhatjuk azt is, hogy a modell két komponense *nem független*, az adatstruktúra megválasztása befolyásolja az algoritmus megválasztását és ez fordítva is igaz.
- Ugyanaz az algoritmus, amely hatékony egyfajta adatstruktúrával, nagyon rossz hatásfokú lehet egy másikkal. Ezért a gyakorlati tervezés általában egy *iteratív, javító, módosító lépéseket is tartalmazó folyamat.*
- A *modellellenőrzés* során a rendszer egy korábban megalkotott modelljéről bizonyítjuk be, hogy teljesülnek-e rá a kívánt tulajdonságok.

Modellellenőrzés

A modellellenőrzés olyan plusz előnyökkel rendelkezik, amelyek már a fejlesztés korai, specifikációs szakaszában is megjelennek.

Ideális esetben:

- a modell határozza meg a designt,
- a szoftvert automatikus kódgenerálással készítjük,
- a validációt pedig modellellenőrzéssel.

A technikák hatékony alkalmazásához azonban tisztában kell lennünk az alkalmazható és alkalmazandó formális módszerekkel, illetve gyakorlatot kell szereznünk az eszközök és technológiák területén is.

A modell végessége

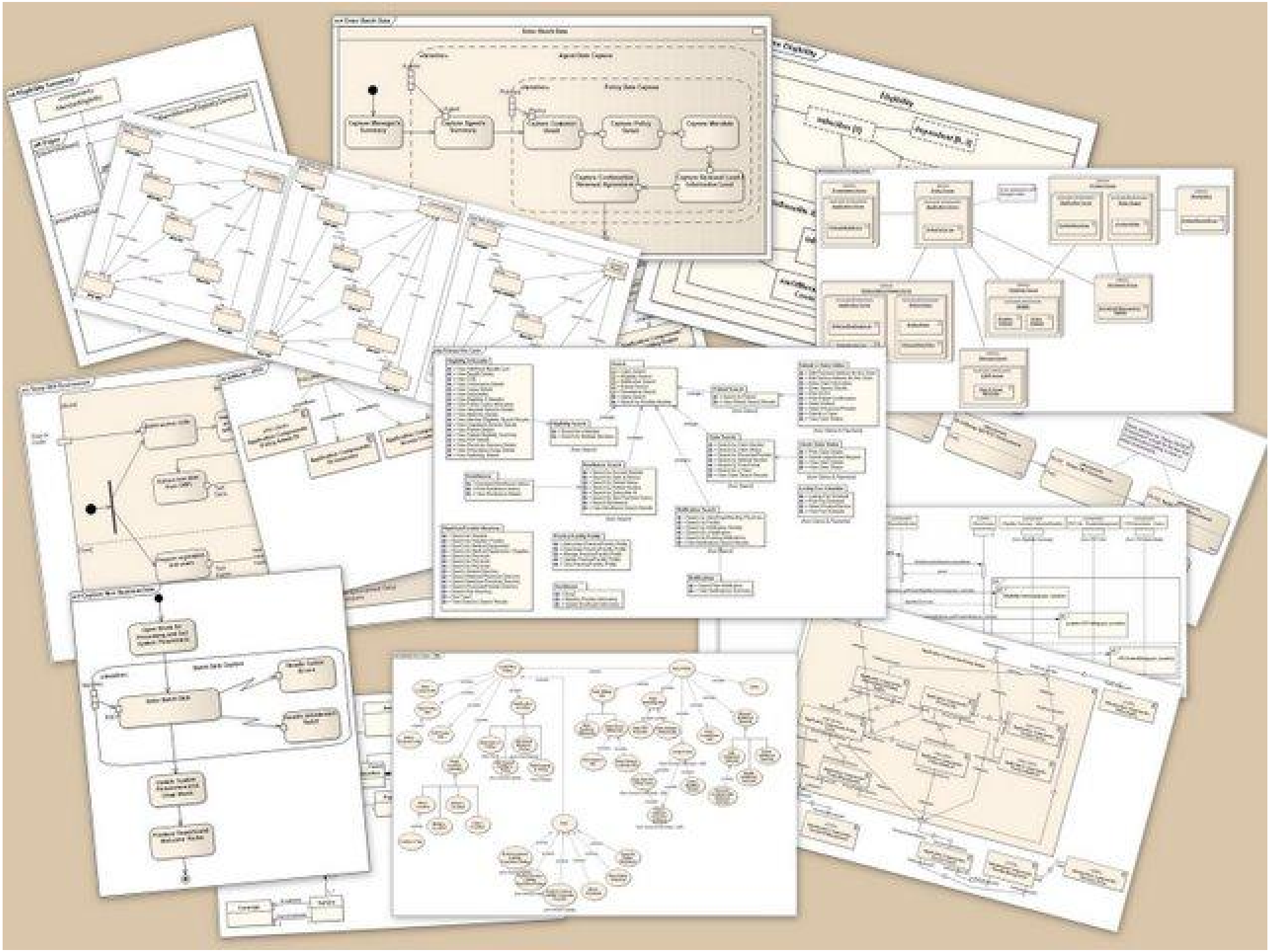
- A modell végessége elvileg garantálja, hogy az algoritmus futása előbb-utóbb véget ér, azonban a gyakorlati életben általában olyan hatalmas állapotterek jönnek létre, amelyek teljes, kimerítő bejárása reménytelen, esélytelen vállalkozás.
- **Kihívás:** a mérhetetlenül nagy állapotterek kezelése.
- Az utóbbi időben két gyakorlati megvalósítás terjedt el:
  - a temporális modellellenőrzés, illetve
  - az automaták formájában történő specifikáció.
- A temporális modell és a véges automaták modellje matematikai módszerekkel egymásnak megfeleltethető, ekvivalenssé tehető.

Temporális modellellenőrzés

- Alapvetően a temporális logikára épít, és a rendszereket mint véges állapotú rendszereket modellezi.
- A temporális logika *az események időbeli tanulmányozására szolgál*.
- A **modális logika** egy változata, ahol az operátorokat arra használják, hogy a tények igazságának idejét jelöljék meg.
- Temporális logikában például ilyen állításokat fogalmazhatunk meg: „p a jövőben mindig, *minden pillanatban* igaz lesz”, „p a jövőben *valamikor* igaz lesz” stb.

Véges automaták (másik megközelítés)

- Az automata ugyanúgy matematikai modell, mint a temporális logika, vagy bármely más formalizmus, azonban rendelkezik azzal az óriási előnnyel, hogy már ránézésre is sok minden leolvasható belőle.
- Egy beszédes UML ábra gyorsabban és több dolgot elárul, mint egy több oldalas magyarázkodás, amely ráadásul sok esetben még félre is értelmezhető.
- (Egységes Modellező Nyelv (Unified Modelling Language - A Rational Software Corporation által kifejlesztett nyelv, amelyet az objektumorientált tervezés adatleírási céljaira fejlesztettek ki)
- Az automata mindig valamilyen meghatározott állapotban van. Ezen állapotok közül mindig egy, és csakis egy érvényes (ekkor azt mondjuk, hogy az automata ebben az állapotban van).



Gondok:

- A modellellenőrzési módszerek legfőbb veszélye *az állapottér robbanása*, azaz a probléma bonyolultságának növelésével az állapotok és állapotátmenetek száma exponenciálisan nő.
- Ma már különböző minimalizáló és redukciós eljárások segítségével akár 10^{120} elérhető állapot is vizsgálható (természetesen automatizált, gépi módszerekkel).

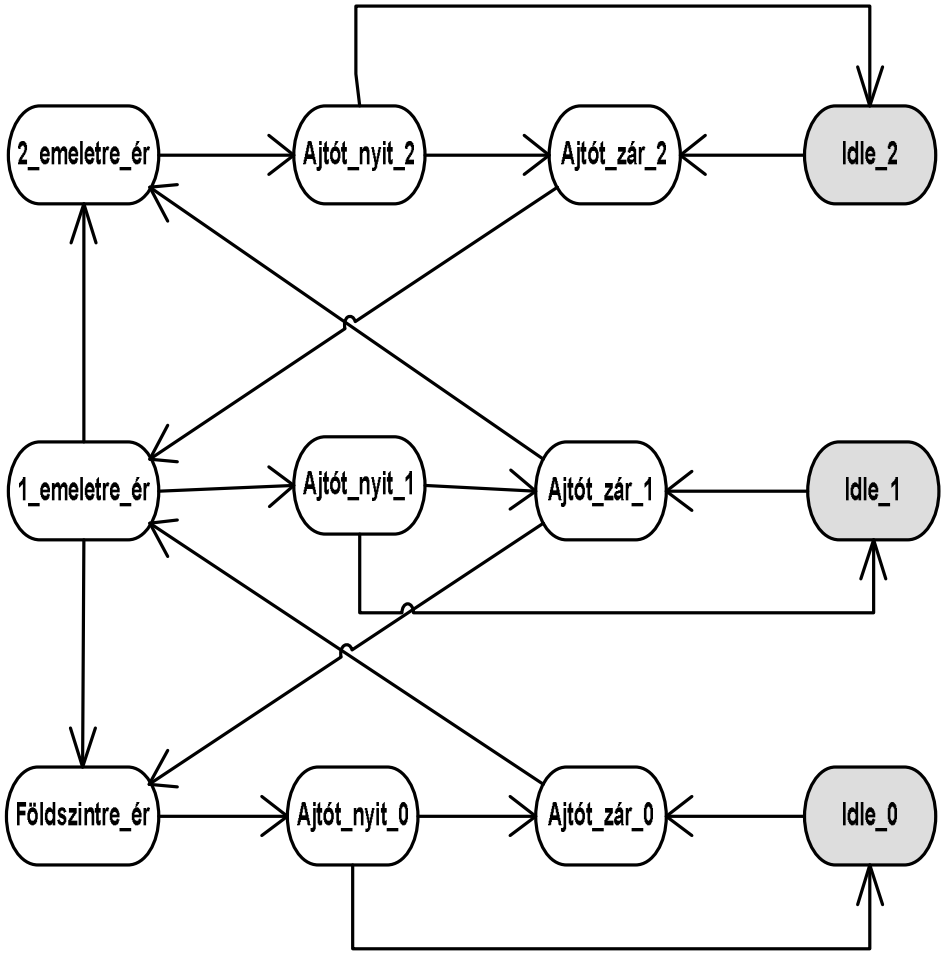
Létjogosultság

Elhíresült példa az IEEE FUTUREBUS szabvány (1986), amelyet különböző formális analíziseknek alávetve kiderült, hogy számos hibát és potenciális kétértelműséget tartalmaz. Sőt, a későbbiekben időanalízissel kibővített módszer még további hibákat is felfedett. Ez volt az első olyan eset, amikor egy nemzetközi szabványról matematikai módszerekkel sikerült bebizonyítani, hogy hibás.

Példa

- Képzeljünk el egy épületet, ahova liftet szeretnénk építeni. A lift alapesetben nyugalmi állapotban van valamelyik emeleten (*Idle*). Ha utasítás érkezik, ajtaja becsukódik, és a lift elindul a megfelelő irányba. Ha menet közben olyan közbülső emeletre ér, ahonnan szintén hívták, akkor ott is megáll.

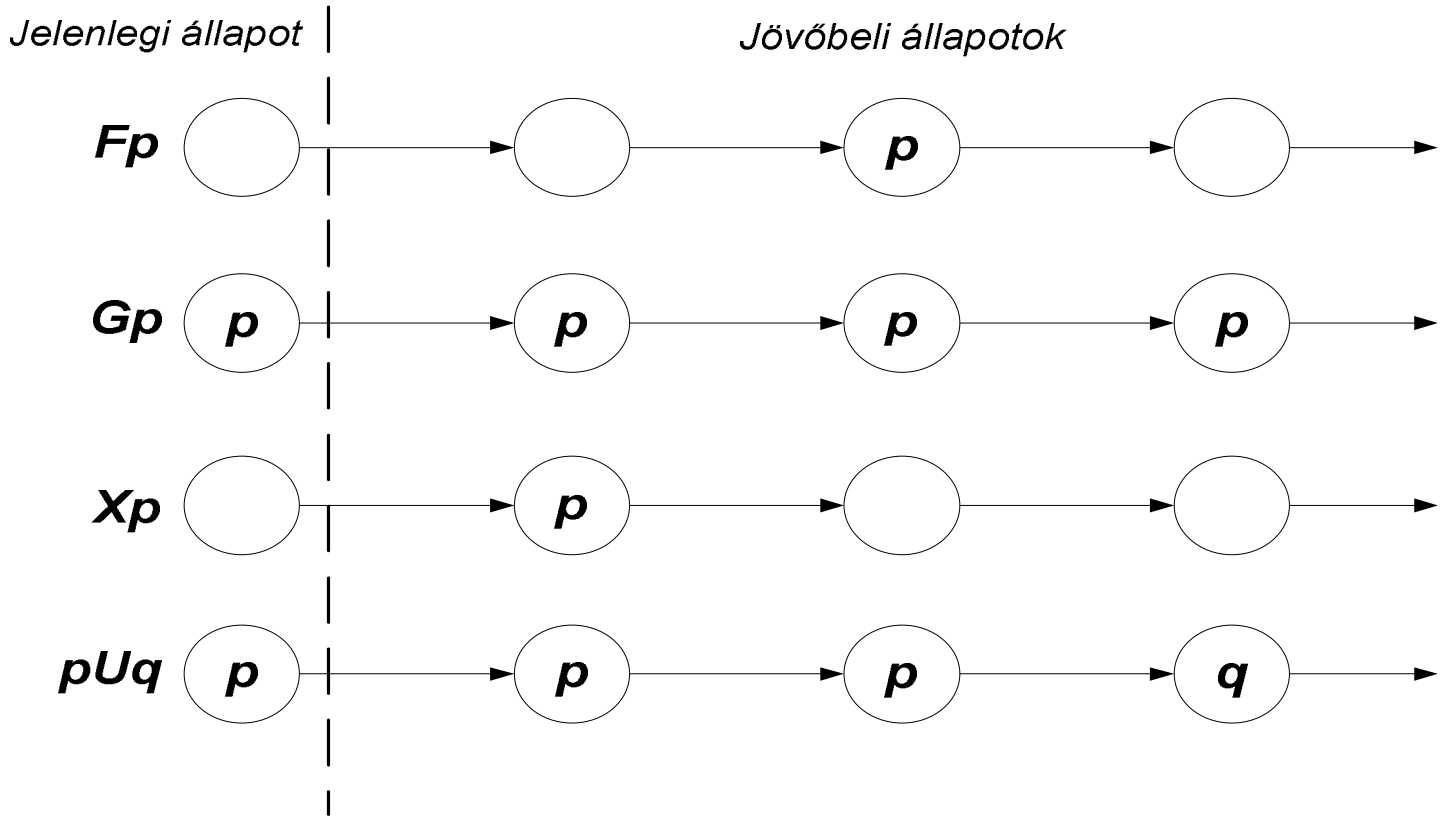
A modellállapot-automatája háromszintes épületet feltételezve (földszint plusz két emelet).



A rendszer a temporális logika nyelvén

- Milyen operátorokat használhatunk az elemi temporális kifejezésekben:

Temporális operátor	***Az operátor jelentése***
Fp	Valamikor *p* (*p* valamikor a jövőben igaz lesz)
Gp	Mindig *p* (*p* mindig igaz)
Xp	Következő *p* (*p* a következő pillanatban igaz lesz)
pUq	*p* amíg *q* (*p* igaz, amíg *q* igaz nem lesz)



G((Ajtót_nyit_1) **è** *(X(Ajtót_zár_1)* s *X(Idle_1))*

Mindig igaz, hogy ha valamikor az első emeleten kinyílik az ajtó, a következő állapot az ajtó bezárása, vagy nyugalom lesz.

G(F(Idle_0) **è** *(Idle_0)U(Ajtót_zár_0))*

Mindig igaz, hogy ha a földszinten valamikor nyugalmi állapotba kerülünk, ott is maradunk egészen addig, amíg az ajtó bezárására (és az indulásra) nem kapunk utasítást).

Miért is jó mindez?

- a rendszerünk tulajdonságait, követelményeit *implementációfüggetlen* módon írhatjuk le;
- olyan rendszereket is vizsgálhatunk a temporális logika segítségével, *amelyek be- és kimenő adatainak kapcsolata nem adható meg egyszerű transzformációként,* hanem az időtényező is lényegi szempont (ilyenek például a folyamatos működésű, reaktív rendszerek, az operációs rendszerek stb.).

Petri-hálók

A Petri-hálók egyidejűleg nyújtanak grafikus és matematikai reprezentációt

- konkurens
- aszinkron
- elosztott
- párhuzamos
- nemdeterminisztikus és/vagy
- sztochasztikus

rendszerek modellezésére.

- Egy rendszer *konkurens*, ha benne egyidejűleg működő, önálló egységek kommunikálnak egymással úgy, hogy ezen egységek egymáshoz képest tetszőleges működési fázisban vannak.
- *Aszinkron*nak nevezzük az eseményvezérelt rendszereket,
- *elosztott*nak pedig azokat, amelyeknél az egyes rendszerelemek között funkcionális tagolódás van, azaz valamilyen megegyezés arról, ki milyen feladatot lásson el a teljes és hatékony működés érdekében.
- A *párhuzamos* rendszerek nagyon hasonlítanak a konkurensekre, lényeges különbség azonban, hogy párhuzamos esetben a rendszerelemek között szoros szinkronizáció áll fenn (konkurens rendszereknél erre semmilyen megkötés nincs).
- Ha egy rendszer *nemdeterminisztikus*, az azt jelenti, hogy egy-egy adott állapotából nem egyértelmű, melyik állapot lesz a következő. Általában adatfüggő, hogy hova lépünk tovább, azonban előfordulhat, hogy a működési módot véletlennel modellezzük.

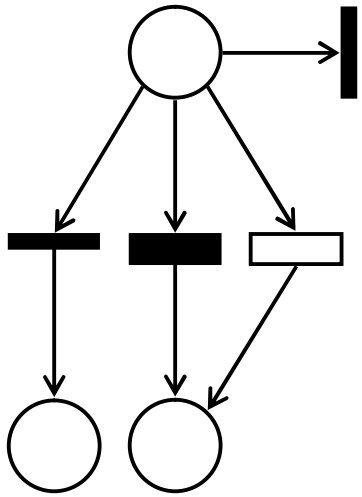
- C.A. Petri a 60-as évek elején publikálta a róla elnevezett módszert.
- Mindent **struktúrával** fejez ki: a vezérlést és az adatszerkezetet egyaránt.
- Ennek a megközelítésnek egyik fő előnye, *hogy minden más ábrázolásmód kiteríthető Pethi-hálóvá*, ugyanakkor már egyszerű feladatok leírása is hatalmas hálót eredményez.
- Összességében a kiforrott matematikai háttér miatt ez a leírásmód rendkívül hatékony eszköz lehet rendszerek analízisére, ha a rendszer modelljét valamely kompaktabb modellezésből automatikusan származtatjuk.

- Egyidejűleg:
  - grafikus
  - matematikai reprezentáció
- Struktúrával fejezi ki:
  - vezérlési struktúra
  - adatstruktúra
- Előnyök/hátrányok:
  + más ábrázolásmódok is kiteríthetőek Petri hálóvá
  - egyszerű feladathoz is nagy Petri háló tartozhat

Petri hálók struktúrája

Strukturálisan: irányított, súlyozott, páros gráf

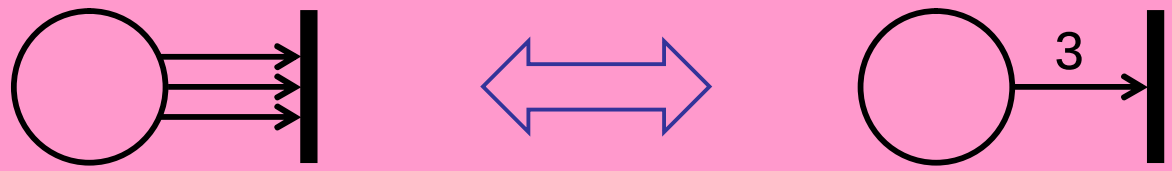
- Két típusú csomópont:
  - hely: $p \in P$ O
  - tranzíció: $t \in T$ □
- Irányított élek (páros gráf):
  - hely $\rightarrow$ tranzíció
  - tranzíció $\rightarrow$ hely
  - $e \in E : (P \times T) \cup (T \times P)$



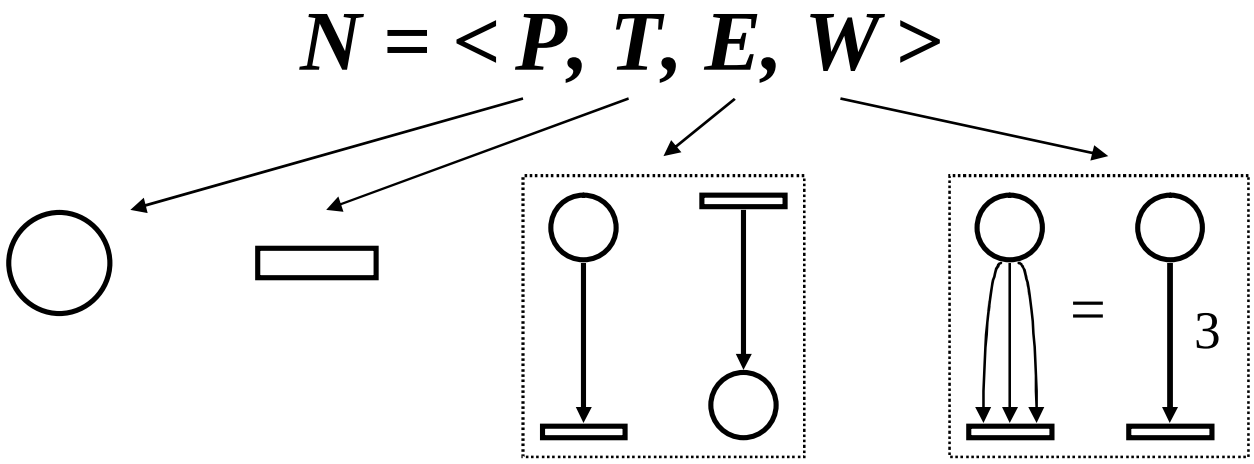
Élek

Élsúly:

- Bármely $e \in E$ élhez $w^*(e) \in \mathbf{N}^+$ súlyt lehet rendelni
- A $w^*(e)$ súlyú e él ugyanaz, mint w_e darab párhuzamos él
- Nem szokás feltüntetni az egyszeres súlyokat



Petri hálók felépítése

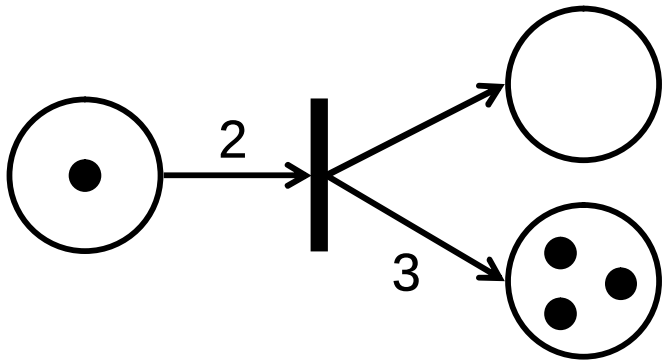


Dinamikus működés: állapotváltozók

Állapot:

- Állapotjelölő: token
  - token jelölése: fekete pötty a hely körébe rajzolva
- Hely állapota: benne levő tokenek száma
- Hálózat állapota: az egyes helyek állapotainak összessége
  - Állapotvektor: a $\pi = |P|$ komponensű M token eloszlás vektor
  - Az m_i komponense a p_i helyen található tokenek száma
    - „p_i –t m_i token jelöli”

$$M = \begin{bmatrix} m_1 \\ \mathbb{M} \\ m_p \end{bmatrix}$$



- Kezdőállapot: $M0$ kezdő token elosztás

Működés

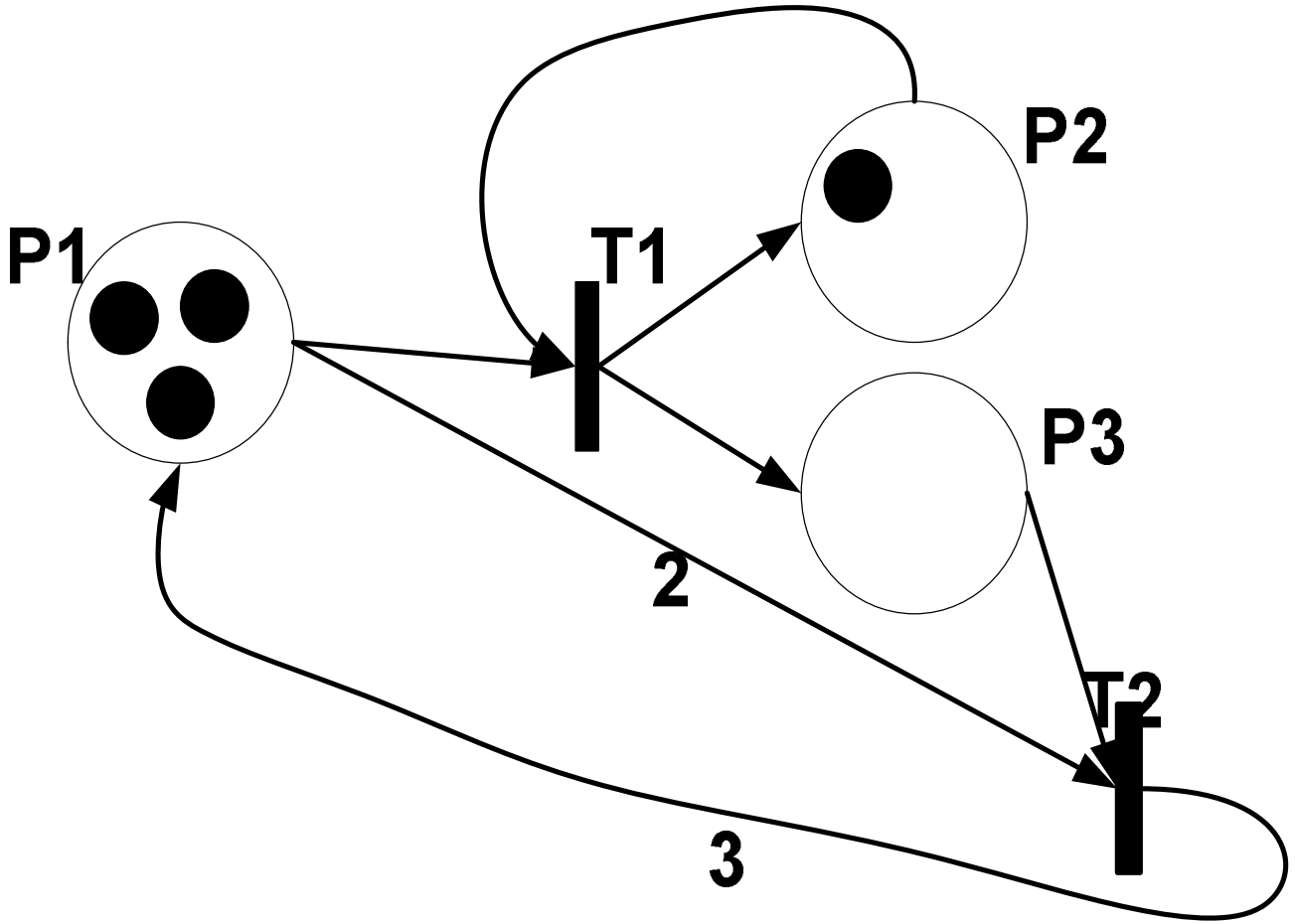
Állapotváltás:

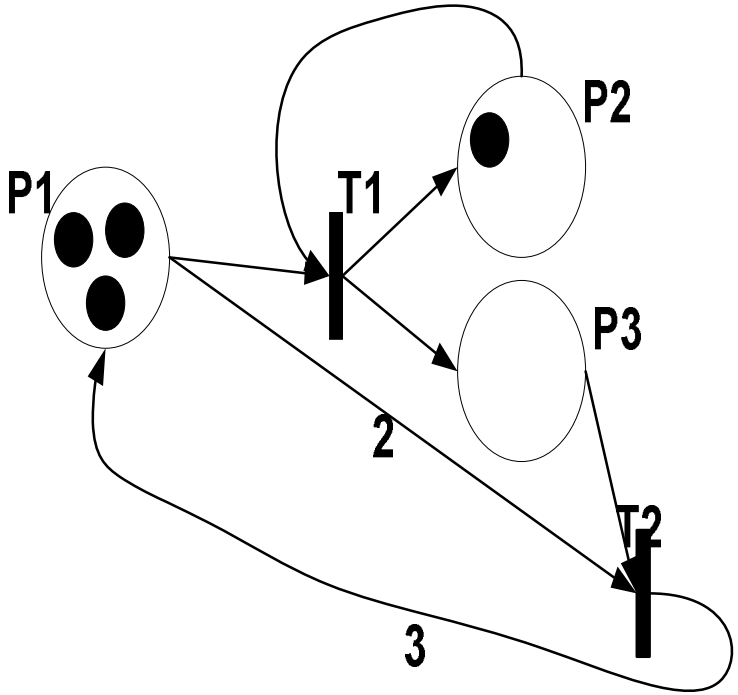
- Állapot megváltozása: tranzíciók „tüzelése”
  - engedélyezettség vizsgálata
  - tüzelés végrehajtása
    - tokenek elvétele a bemeneti helyekről
    - tokenek kirakása a kimeneti helyekre
  - megváltozott token eloszlás vektor: új állapot

Speciális csomópontok

- a forrás- és a nyelő tranzíció
- A forrásnak nincs bemenete, és mindig képes tüzelni
- a nyelő tranzíciónak pedig nincs kimenete, így a tüzelés során a hozzá érkező tokeneket „elnyeli”.

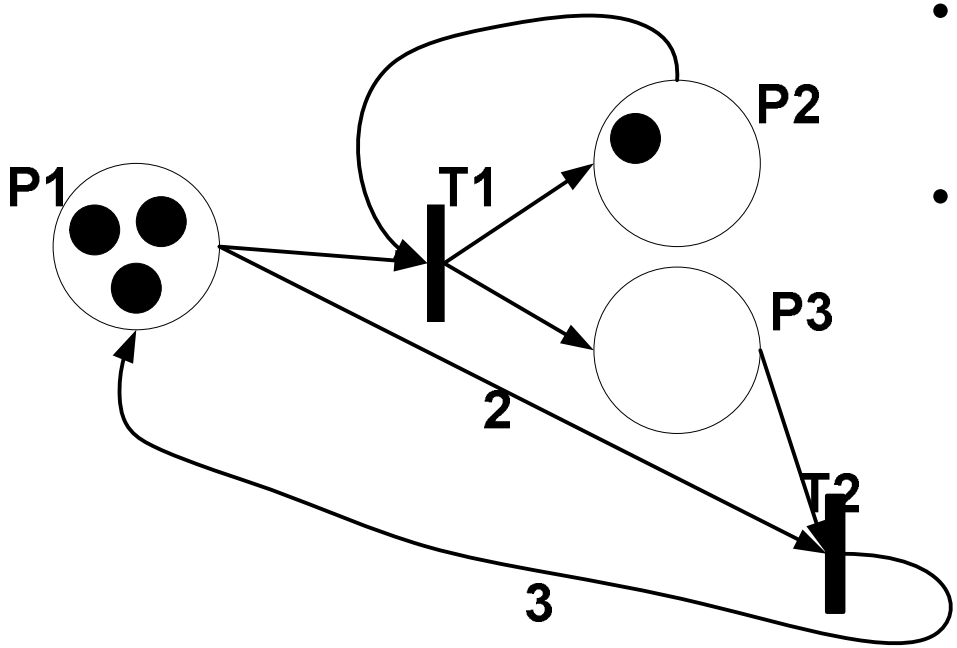
Példa





- egy egyszerű Petri-háló kezdeti állapotát (tokeneloszlását) láthatjuk:
- A *P1* állapotban 3,
- a *P2*-ben 1,
- *P3*-ban 0 token van.
- A *T1* tranzíciónak két bemenő (*P1*-ből és *P2*-ből), és két kimenő (*P2*-be és *P3*-ba) éle van.
- Két élnek van 1-től különböző súlya: a *P1* **à** *T2* él 2, a *T2* **à** *P1* él 3 súlyú.

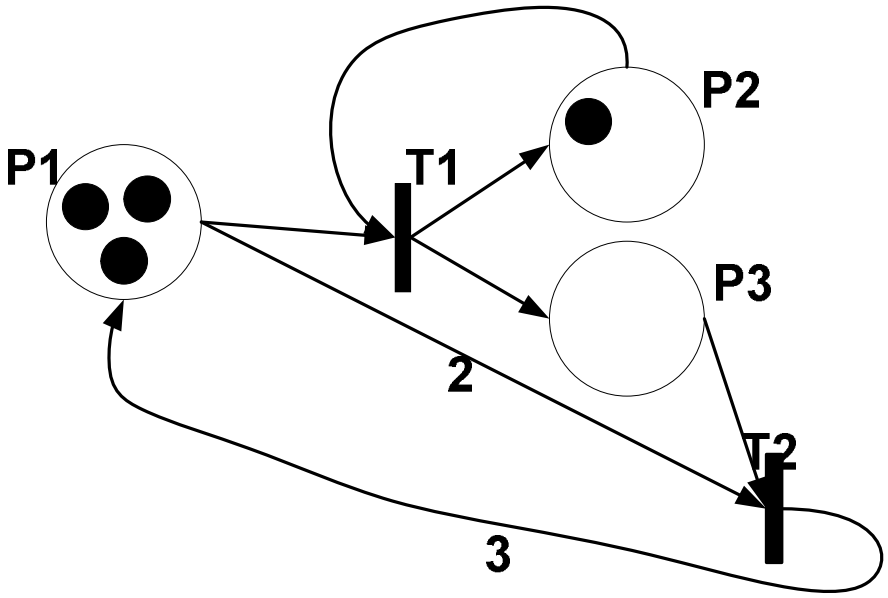
- Mit is jelent mindez? Petri-hálók esetében állapotátmenet helyett *tüzelés*ről beszélünk: ez az a folyamat, amely során a tokenek ide-oda vándorolnak a hálón belül.
- Egy tranzíció akkor tüzel*het*, ha az összes bemenő éléhez csatlakozó helyen van legalább annyi token, amennyi az adott él súlya.

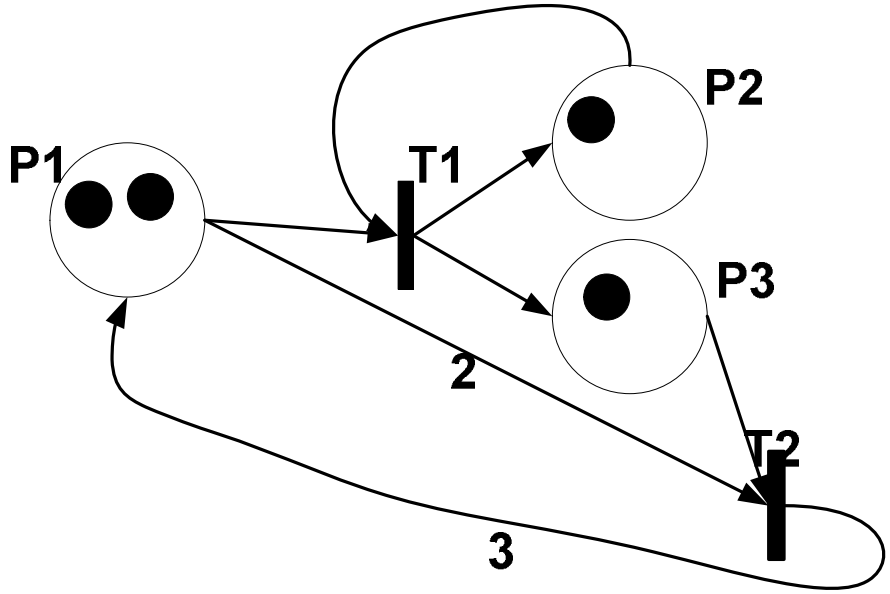


- *T1* tüzelhet, mert *P1*-ben is, *P2*-ben is van 1-1 token,
- *T2* azonban nem, mert *P1*-ben van ugyan 2 token, de *P3*-ban nincs egy sem.
- A tüzelés során a tranzíció a bemenő éleihez csatlakozó helyekről (az őseitől) elveszi az élnek megfelelő számú tokent, a kimenő élek végpontjain álló helyekhez (utódaihoz) pedig hozzáad az él súlyának megfelelő számút.
- A példában a *T2* tranzíció a *P3* helyről 1, a *P1* helyről 2 tokent vesz el a tüzelés során.

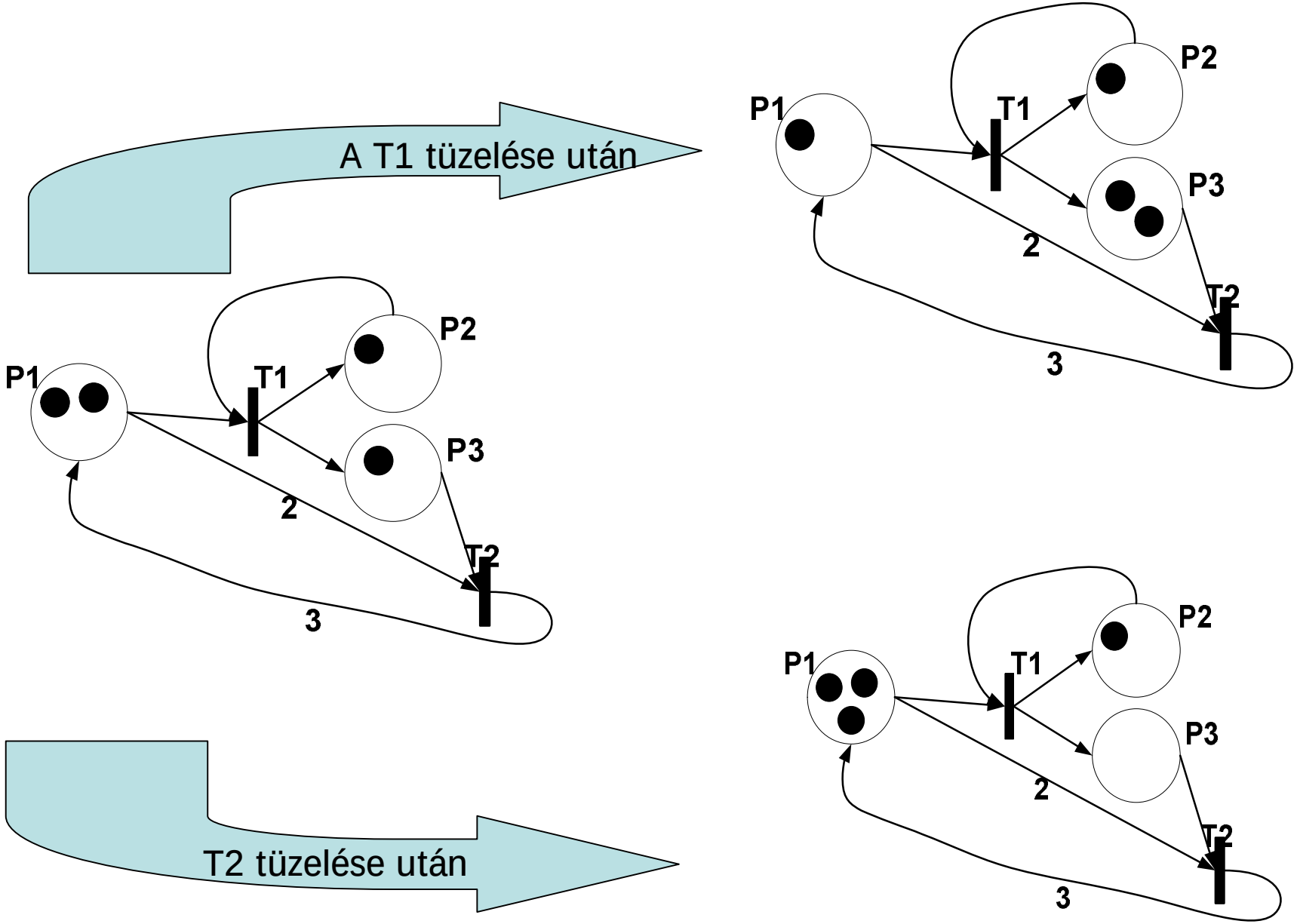
Most egyértelmű, hogy a következő tüzelő tranzíció a T1 lesz, de mi történik, ha egyszerre több tranzíció is tüzelhetővé válik? Ebben az esetben is egyetlen tranzíció tüzel a következő alkalommal (a következő logikai időpillanatban), de hogy melyik, az előre teljesen kiszámíthatatlan (a Petri-háló nemdeterminisztikus volta miatt).

Hogyan néz ki a fenti háló a T1 tranzíció tüzelése után?

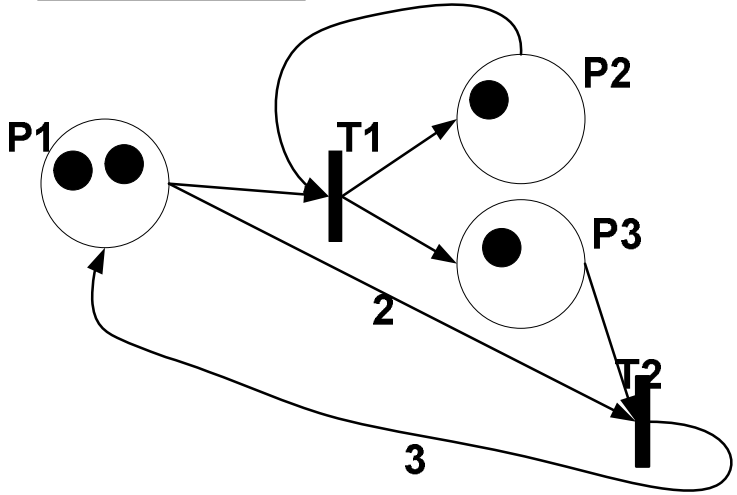


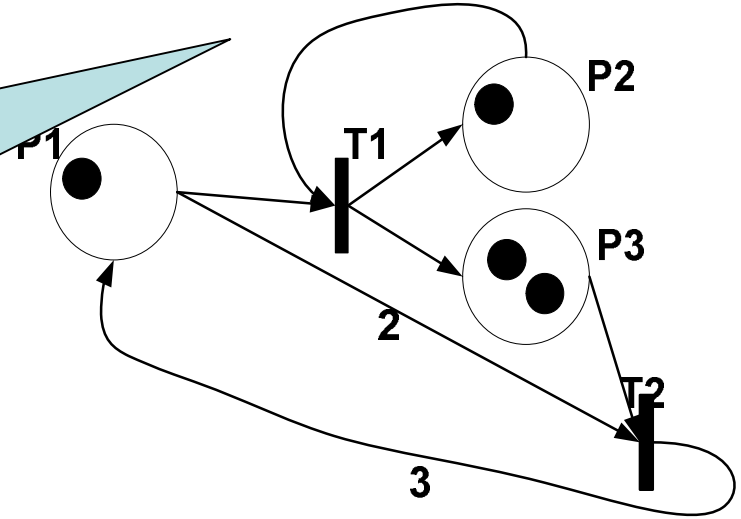


Jól látható, hogy most már mind a T1, mind a T2 állapot tüzelhető.

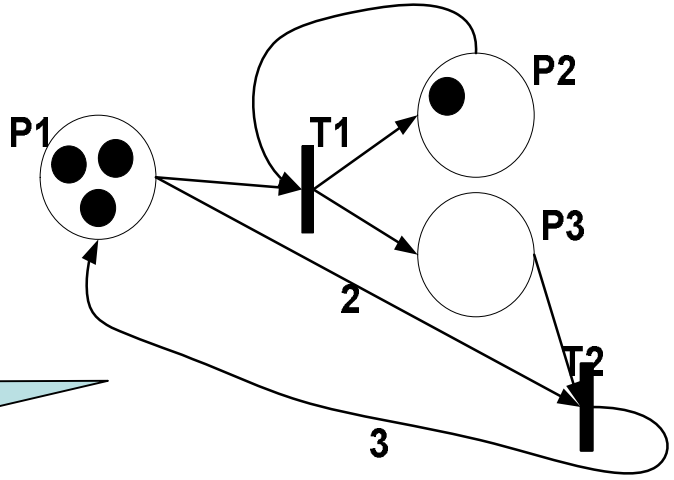

A T1 tüzelése után
P1
T1
P2
P3
2
T2
3
P1
T1
P2
P3
2
T2
3
T2 tüzelése után
P1
T1
P2
P3
2
T2
3

-*T2* tranzíció ismét nem képes tüzelésre
-ha a T1 tranzíció még egy alkalommal tüzel, akkor a *P1* helyen nem marad több token, a háló holtpontba (deadlock) kerül





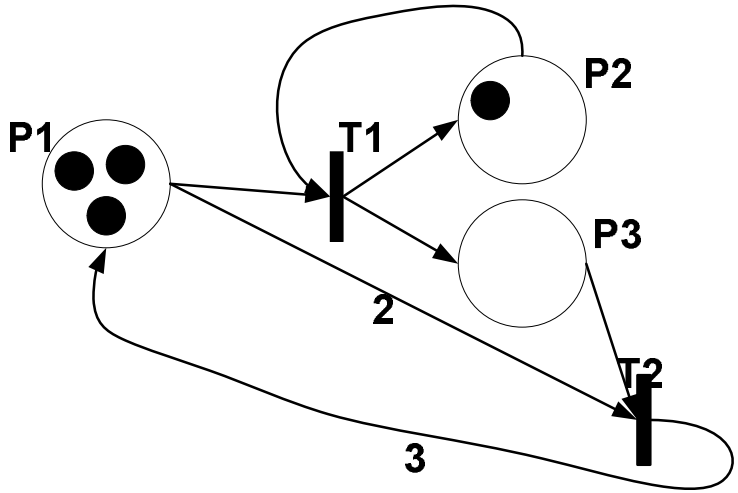
megegyezik a kiinduló tokeneloszlással



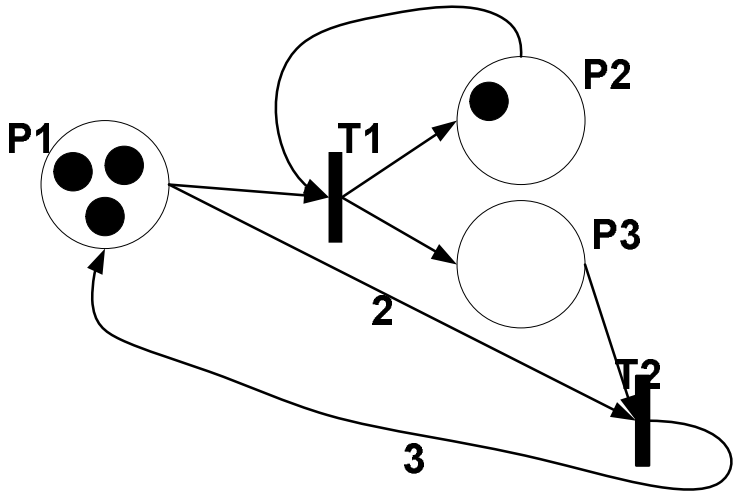
Matematikai formalizmussal:

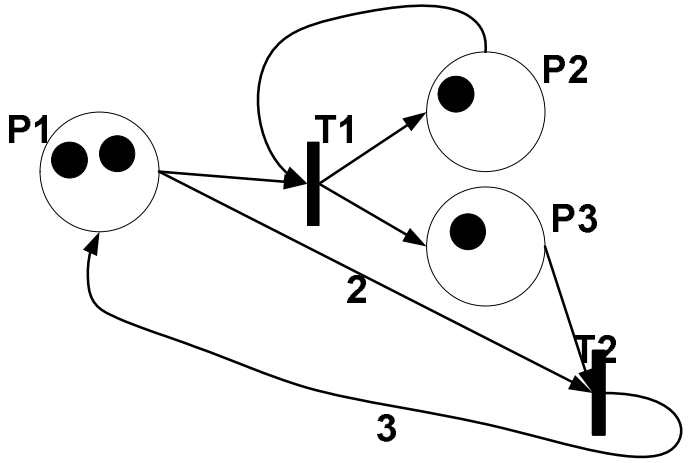
- Egy adott tokeneloszlást az az M vektor jelöl, amelynek dimenziója a P helyek száma (π=|P|), elemei pedig az adott sorszámú helyen az adott pillanatban található tokenek száma.
- Az előzőekben tekintett kezdeti tokeneloszlás leírása tehát ily módon

(π =3):



$$M_0 = \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix}$$





$$M_0 = \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix}$$

$$M_{T1} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Összefoglaló

Petri háló:

- Nemdeterminisztikus véges automata
- Állapotvektor: token eloszlás vektor
- Állapotátmeneti függvény: tranzíciók

Felépítés:

- egy-egy hely egy-egy logikai feltétel
- Petri háló struktúrája követi a feladat logikai dekompozícióját

Szintakszis összefoglalása

Petri-háló	$PN = (P, T, E, W, M_0)$
Helyek	$P = \{p_1, p_2, \cdots p_\pi\}$
Tranzíciók	$T = \{t_1, t_2, \cdots t_\tau\}$
	$P \cap T = \emptyset$
Élek	$E \subseteq (P \times T) \cup (T \times P)$
Súlyfüggvény	$w^* : E \longrightarrow \mathbb{N}^+$
Kezdőállapot	$M_0 : P \longrightarrow \mathbb{N}$
PN struktúra	$N = (P, T, E, W)$
PN adott kezdőállapottal	(N, M_0)

Topológia

$n \in (P \cup T)$ csomópont $\bullet n$ ősei és $n \bullet$ utódai:

- $t \in T$ ősei a bemeneti helyei: $\bullet t = \{p \mid (p,t) \in E\}$
- $t \in T$ utódai a kimeneti helyei: $t \bullet = \{p \mid (t,p) \in E\}$
- $p \in P$ ősei a bemeneti tranzíciói: $\bullet p = \{t \mid (t,p) \in E\}$
- $p \in P$ utódai a kimeneti tranzíciói: $p \bullet = \{t \mid (p,t) \in E\}$

Topológia

Csomópontok $P' \subseteq P$ ill. tranzíciók $T' \subseteq T$ részhalmazára:

$$\bullet P' = \bigcup_{p \in P'} \bullet p$$

$$P' \bullet = \bigcup_{p \in P'} p \bullet$$

$$\bullet T' = \bigcup_{t \in T'} \bullet t$$

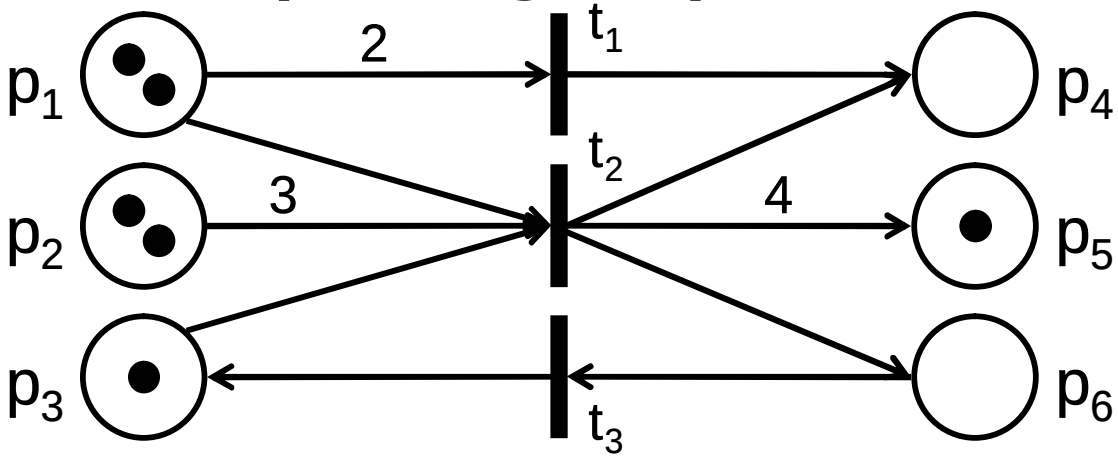
$$T' \bullet = \bigcup_{t \in T'} t \bullet$$

Speciális csomópontok

Forrás ill. nyelő csomópontok

- $t \in T$ forrás (nyelő) tranzíció:
  - Bemenő (kimenő) hely nélküli ($\bullet t = \emptyset$ illetve $t \bullet = \emptyset$)
  - Forrás tranzíció minden esetben tud tüzelni
- PN **tiszta**, ha nincsenek önhurkai, azaz
  - $\forall t \in T : \bullet t \cap t \bullet = \emptyset$

Topológia példa



$\mathbf{I}\, p_1 = \varnothing$
$\mathbf{I}\, p_2 = \varnothing$
$\mathbf{I}\, p_3 = \{t_3\}$
$\mathbf{I}\, p_4 = \{t_1, t_2\}$
$\mathbf{I}\, p_5 = \{t_2\}$
$\mathbf{I}\, p_6 = \{t_2\}$

$p_1\, \mathbf{I} = \{t_1, t_2\}$
$p_2\, \mathbf{I} = \{t_2\}$
$p_3\, \mathbf{I} = \{t_2\}$
$p_4\, \mathbf{I} = \varnothing$
$p_5\, \mathbf{I} = \varnothing$
$p_6\, \mathbf{I} = \{t_3\}$

$\mathbf{I}\, t_1 = \{p_1\}$
$\mathbf{I}\, t_2 = \{p_1, p_2, p_3\}$
$\mathbf{I}\, t_3 = \{p_6\}$

$t_1\, \mathbf{I} = \{p_4\}$
$t_2\, \mathbf{I} = \{p_4, p_5, p_6\}$
$t_3\, \mathbf{I} = \{p_3\}$

Dinamikus viselkedés

Petri hálók „működésének” egy lépése:

- Állapot megváltozása: tranzíciók „tüzelése”
  - korábbi állapot: kezdeti token eloszlás vektor
  - tüzelés végrehajtása
    1. engedélyezettség vizsgálata
    2. tokenek elvétele a bemeneti helyekről
    3. tokenek kirakása a kimeneti helyekre
  - új állapot: megváltozott token eloszlás vektor

Tüzelés feltétele

- Ha egy $t \in T$ tranzíció minden bemeneti helyét legalább $w^-(p, t)$ token jelöli:
  - $w^-(p, t)$ a p-ből t-be vezető $e = (p, t)$ él súlya
  - $\Rightarrow$ a tranzíció tüzelése **engedélyezett**, ha

$$\forall p \in \bullet t : m_p \geq w^-(p,t)$$

Állapotátmenet

Tüzelés végrehajtása:

- Engedélyezett tranzíció tetszése szerint tüzelhet vagy nem
  - “fire at will”
- Több tranzíció engedélyezett: konfliktus
  - egy lépésben csak egy engedélyezett tranzíció tüzelhet
  - konfliktusfeloldás véletlen választással
- Þ **Nemdeterminisztikus működés**

A tranzíció tüzelése

- elvesz $w^-(p, t)$ darab tokent a $p \in {}^\bullet t$ bemeneti helyekről
  - $w^-(p, t)$ a $p \to t$ él súlya
- elhelyez $w^+(t, p)$ darab tokent a $p \in t^\bullet$ kimeneti helyekre
  - $w^+(t, p)$ a $t \to p$ él súlya

Nemdeterminizmus és időzítés

Tetszés szerinti tüzelés jelentése:

- implicit időfogalom
- nincs időskála
- a tüzelés a [0, ∞) időintervallumban valahol megtörténhet

A tüzelésekhez tetszőleges konkrét időértéket rendelve:

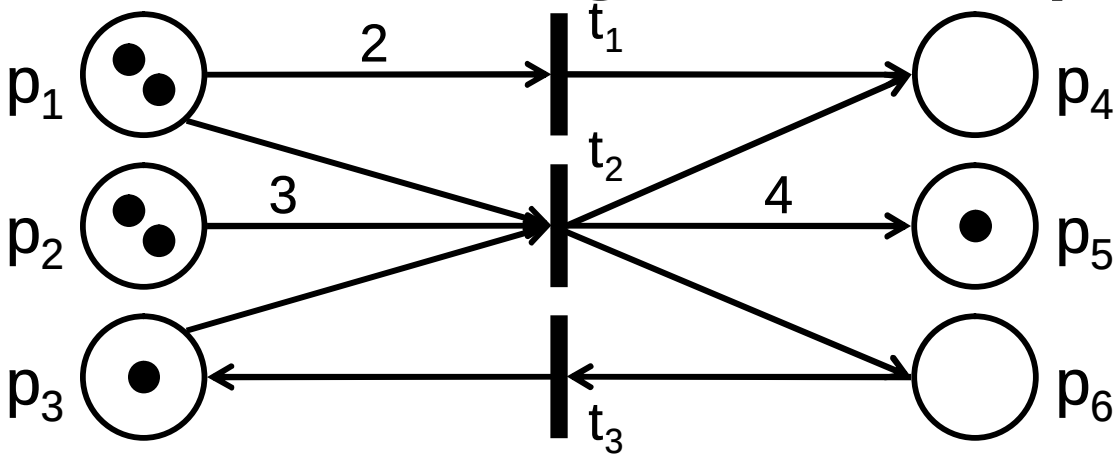
- az azonos struktúrájú és kezdőállapotú nem-determinisztikus időzítetlen Petri háló annak minden lehetséges tüzelési szekvenciáját lefedi.

Szomszédossági mátrix

- Súlyozott szomszédossági mátrix: **W** = [w(t, p)]
- Dimenziója: $\tau \times \pi = |T| \times |P|$
- Ha t tüzel, mennyit változik a p-beli tokenszám:

$$w(t, p) = \begin{cases} 0 & \text{ha } (t, p) \notin E \text{ és } (p, t) \notin E \\ w^{+}(t, p) - w^{-}(p, t) & \text{ha } (t, p) \in E \text{ vagy } (p, t) \in E \end{cases}$$

Szomszédossági mátrix példa



$$W = \begin{bmatrix} -2 & 0 & 0 & 1 & 0 & 0 \\ -1 & -3 & -1 & 1 & 4 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$W^- = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W^+ = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$