(Extra: Formális módszerek szerepe)

Matematikai technikák, elsősorban diszkrét matematika és matematikai logika használata arra, hogy elkészítsük és ellenőrizzük hardver és szoftver rendszerek specifikációját, terveit (modelljeit), implementációját (viselkedését), dokumentációját.

Formalizálás célja: matematikai precizitású megadás.
Tervek (modellek, tervezői döntések) és követelmények (elvárt tulajdonságok)
Formális nyelv felépítése: szintaxis (nyelvi elemek és kapcsolatuk) és szemantika (mit jelentenek)

A formális leírás *tipikusan absztrakt* (implementáció-független), *sokféle szempontú* lehet (struktúra, funkció, teljesítmény, biztonság, ….)

Előnyei:

- Tervek, algoritmusok precíz leírása: egyértelmű, ellenőrizhető
- Tervek helyességének ellenőrzése: megállapítható a helyesség, vagy felderíthetőek a problémás esetek, korán javítható a hiba!
- Automatikusan feldolgozható: szintaxis ellenőrzés, formális verifikáció, alkalmas kódgeneráláshoz

Formális modell használata:

- **Végrehajtás:** szimulációval
- **Ellenőrzés:** formális verifikáció, konzisztenciát, teljességet, zártságot, elvárt tulajdonságok teljesülését, modellfinomítás után helyesség megmaradását
- **Szintézis:** modellből szoftvert (programkód, konfiguráció) vagy hardver implementációt generálni

Formális verifikáció: mérnöki formális modell + formalizált követelmények + automatikus modellellenőrző => követelmények teljesülnek, vagy nem

Validáció kell verifikáció mellett is. Felhasználói elvárásoknak megfelel-e a kész rendszer. Verifikáció objektív folyamat, formalizálható, de validációnál szubjektív elvárások is lehetnek. Verifikációnál a tervek hibái deríthetők fel, validációnál a követelmények hibái is.

Formális módszerek nehézségei:

- Valósághű modellezést nehezíti az ismeretek hiánya
- Felhasználónak ismernie kell a modellezési nyelvet
- Ismerni kell az ellenőrzés és szintézis korlátait, kézi beavatkozásra lehet szükség
- Nagy állapottér nem kezelhető a meglévő erőforrásokkal

Y fejlesztési modell: V-modell átalakítása úgy, hogy a tervezés-verifikálás nagyobb hangsúlyt kapjon az implementáció előtt

1. Alapszintű formalizmusok: Kripke-struktúra, LTS, KTS, időzített automata

Kripke-struktúra: Állapotok tulajdonságait fejezzük ki, címkézés atomi kijelentésekkel. Egy állapothoz sok címke rendelhető. Alkalmazás: Viselkedés, algoritmus leírása.

Pl. diák tanul, vizsgázik, pihen állapotok, köztük üres nyilak.

KS = (**S,R,L**) és **AP**, ahol

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

S = {s1,s2,s3, … sn} állapotok halmaza

R ⊆ **S** x **S**: állapotátmeneti reláció

L: **S** -> 2^{AP} állapotok címkézése atomi kijelentésekkel.

LTS, Címkézett tranzíciós rendszer: Állapotátmenetek tulajdonságait fejezzük ki, címkézés akciókkal. Egy átmeneten csak egy akció. Alkalmazás: Kommunikáció, protokollok modellezése.

Pl. ital automata feladat, ahol az állapotok üresek, nyilakon: pénz, kávé, tea.

LTS = (**S, Act, ->**), ahol

S = {s1,s2,s3, … sn} állapotok halmaza

Act = {a,b,c,…} akciók (címkék) halmaza

-> ⊆ **S** x **Act** x **S**: címkézett állapotátmenetek, állapotátmeneti reláció. (Jelölés $s_1 \xrightarrow{a} s_2$)

KTS, Kripke tranzíciós rendszer: Állapotok és átmenetek tulajdonságait is kifejezzük: címkézés atomi kijelentésekkel és akciókkal. Egy állapothoz sok címke rendelhető, egy átmenethez egy címke rendelhető.

Pl. ital automatás feladat: Start állapot, utána pénz akció, Választ állapot, utána kávé vagy tea akció, végül Stop állapot

KTS = (**S, ->, L**) és **AP**, **Act**.

Időzített automata: Véges állapotú automata (FSM) kiterjesztése óraváltozókkal, valós idejű viselkedést és idő függvényében változó viselkedést modellezve. Célja: állapot alapú viselkedés modellezése. Nyelvi kiterjesztések:

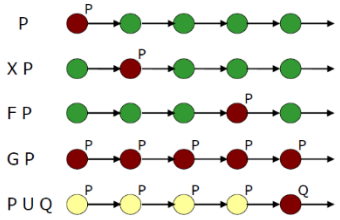
- egész értékű változók használata állapotátmeneteken: *őrfeltétel*ben (átmenet tüzelhet, ha igaz) vagy *akció*ként (tüzeléskor végrehajtódik, változónak *értékadás*).
- *Óraváltozók*: valós idejű viselkedés: állapotokban idő telik, relatív időmérés (reset), viselkedés időtől függő. Állapotátmenetekben (akció, őrfeltétel) és állapotokban (*állapot invariáns*: állapotban lehet, míg igaz)
- *Szinkronizált akciók*: együttműködő automaták modellezése, szinkronizáció, együttlépő átmenetek (szinkron kommunikáció, küldő vár a fogadóra). *Csatornák* definiálásával, operátorok ! és ?. *Broadcast* csatorna is létrehozható, ott a küldő nem vár a fogadóra. *Urgent* csatorna késleltetés nélküli szinkronizációhoz.
- *Urgent állapot*: nem telhet idő az állapotban
- *Committed állapot*: átmenetek egybefogása, bemenő és kimenő átmenet egy atomi művelet, nem ékelődhet közbe más automata átmenete.
- Véletlen választás: *select* megadható egy változóval és típussal, minden lehetséges választást bejár az ellenőrzéskor

2. Lineáris idejű temporális logikák: A PLTL elemkészlete, formális szintakszisa és szemantikája. A PLTL kiterjesztése LTS-re

Temporális logika: Formális rendszer a kijelentések igazságának logikai időbeli változásának vizsgálatára.
Cél: állapottér vizsgálata.

PLTL, Lineáris idejű temporális logika: modell egy-egy lefutását tekintjük. Minden állapotnak van rákövetkezője. Logikai idő egy idővonal mentén (állapotsorozat).
Kifejezések konstruálása:

- Atomi kijelentések (**AP** elemei)
- Boole logikai operátorok: és ^, vagy ˅, negálás ˥, implikáció =>
- Temporális operátorok
  - F: Future, egy elérhető állapotban igaz lesz.
  - G: Globally, minden elérhető állapotban igaz lesz.
  - X: neXt: következő állapotban igaz lesz.
  - U: Until: pl. p U q: egy elérhető állapotban igaz lesz q, addig minden állapotban igaz p.



PLTL formális szemantikája: Szemantikája megadja, hogy mikor igaz egy adott útvonalon egy adott PLTL kifejezés
M = (**S, R, L**) Kripke-struktúra
π = (s_0,s_1,s_2,...) **M** egy útvonala, ahol s_0 a kezdőállapot, és ∀i >= 0: (s_i, s_{i+1}) eleme **R**
π^i = (s_i,s_{i+1},s_{i+2},...) a π útvonal szuffixe i-től
M, π |= p az M modellben a π útvonalon igaz p

PLTL formális szintaxisa: kifejezések alábbi szabályokkal képezhetők

- **L1**: Minden P atomi kijelentés egy kifejezés

M, π |= P ⇔ P ϵ L(s_0)

- **L2**: Ha p és q kifejezések, akkor p ^ q illetve ˥ p is

M, π |= p ^ q ⇔ M, π |= p és M, π |= q
M, π |= ˥ p ⇔ nem igaz hogy M, π |= p

- **L3**: Ha p és q kifejezések, akkor p U q és X p is

M, π |= p U q ⇔ ∃j>=0: (π^j |= q és ∀0<=k<j: π^k |= p)
M, π |= X p ⇔ π^1 |= p

Operátorok precedenciája: ≡, =>, ˅, ^, ˥, (X,U)
Kimaradt operátorok előállíthatóak:

- p ˅ q: nem igaz hogy nem p ^ nem q
- p => q: nem p vagy q
- p ≡ q: p => q ^ q => p
- F p: true U p
- G p: nem igaz hogy F(nem p)

Kibővítés LTS-re:

- π = (s_0,a_1, s_1, a_2, s_2, a_3...) útvonal az akciókkal együtt adható meg
- **L1***: Ha a egy akció, akkor (a) egy PTLT kifejezés

M, π |= (a) ⇔ a = a_1 az első akció π-ben

3. Elágazó idejű temporális logikák: A CTL és a CTL* elemkészlete, formális szintakszisa és szemantikája. A PLTL, a CTL és a CTL* temporális logikák kifejezőerejének összehasonlítása.

Temporális logika: Formális rendszer a kijelentések igazságának logikai időbeli változásának vizsgálatára.
Cél: állapottér vizsgálata

CTL, Elágazó idejű temporális logika*:* modell minden lehetséges lefutását tekintjük. Állapotnak több rákövetkezője lehet. Logikai idő elágazó idővonalat mentén (számítási fa).
Egy-egy állapotban előírható, hogy az útvonalakra vonatkozó p követelmény hány onnan kiinduló útvonal mentén teljesüljön: E *egzisztenciális* operátor -> létezzen legalább egy útvonal, és A *univerzális* operátor -> minden útvonalon legyen igaz

CTL* vs **CTL**: Útvonal kvantorok (A, E) és útvonalakon értelmezett temporális operátorok (X,F,G,U) alkotják, de *CTL a szigorúbb*, útvonalakon értelmezett kvantorokat mindig meg kell előzze útvonal kvantor, és útvonalakon értelmezett operátorok nem kombinálhatók!
CTL*-ban ilyen megkötés nincs.

CTL* formális szemantikája: PLTL szemantikával megegyezik, kibővítve azzal, hogy
M, s |= p az **M** modellben az **s** állapotban igaz p

CTL* formális szintaxisa: Érvényes CTL* kifejezések az alábbi szabályok alapján generált állapot-kifejezések

Állapot-kifejezések:

- **S1**: Minden P atomi kijelentés egy állapot-kifejezés

M, s |= P ⇔ P ∈ L(s_0)

- **S2**: Ha p és q állapot-kifejezések, akkor p ^ q illetve ¬ p is

M, s |= p ^ q ⇔ M, s |= p és M, s |= q
M, s |= ¬ p ⇔ nem igaz hogy M, s |= p

- **S3**: Ha p útvonal-kifejezés, akkor E p és A p állapot-kifejezések

M, s |= E p ⇔ ∃π = (s_0,s_1,s_2,…) útvonal M-ben s=s_0 mellett, hogy M, π |= p
M, s |= A p ⇔ ∀π = (s_0,s_1,s_2,…) útvonalra M-ben, ahol s=s_0 , igaz, hogy M, π |= p

Útvonal-kifejezések:

- **P1**: Minden p állapot-kifejezés útvonal-kifejezés

M, π |= p ⇔ M, s_0 |= p

- **P2**: Ha p és q útvonal-kifejezések, akkor p ^ q illetve ¬ p is

M, π |= p ^ q ⇔ M, π |= p és M, π |= q
M, π |= ¬ p ⇔ nem igaz hogy M, π |= p

- **P3**: Ha p és q útvonal-kifejezések, akkor p U q és X p is

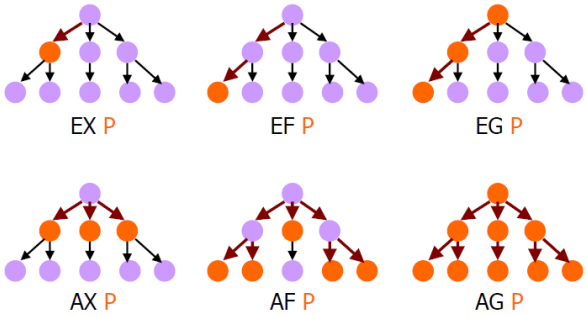
M, π |= p U q ⇔ ∃j>=0: (π^i |= q és ∀0<=k<j: π^k |= p)
M, π |= X p ⇔ π^1 |= p

CTL formális szintaxisa: Állapot-kifejezések (S1, S2, S3) megegyezik CTL*-gal.

Útvonal-kifejezés csak egy van az előző 3 helyett:

- **P0**: Ha p és q állapot-kifejezések, akkor X p és p U q útvonal-kifejezések
  M, π |= X p ⇔ M, s_1 |= p
  M, π |= p U q ⇔ ∃j>=0: (s_j|= q és ∀0<=k<j: s_k |= p)

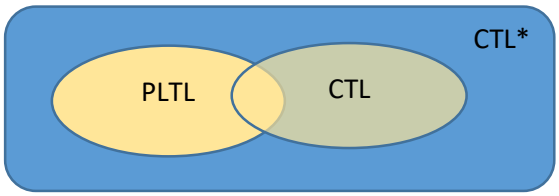
Útvonal-kifejezések nem kombinálhatók, és csak S3 szabály használhatja őket.
X p és p U q útvonal-kifejezések elé csak valamilyen útvonal kvantor kerülhet.



Kimaradt CTL operátorok előállíthatóak:

- EF p: E(true U p)
- AF p: A(true U p), univerzálisan igaz (minden útvonalra), hogy valamikor igaz lesz
- EG p: nem igaz hogy AF(nem p), létezik olyan útvonal ahol minden állapotra igaz
- AG p: nem igaz hogy EF(nem p)

Temporális logikák kifejező képessége: CTL* kifejező ereje a legnagyobb. PLTL-nek és CTL-nek van metszete, és mindketten CTL* részhalmazi.



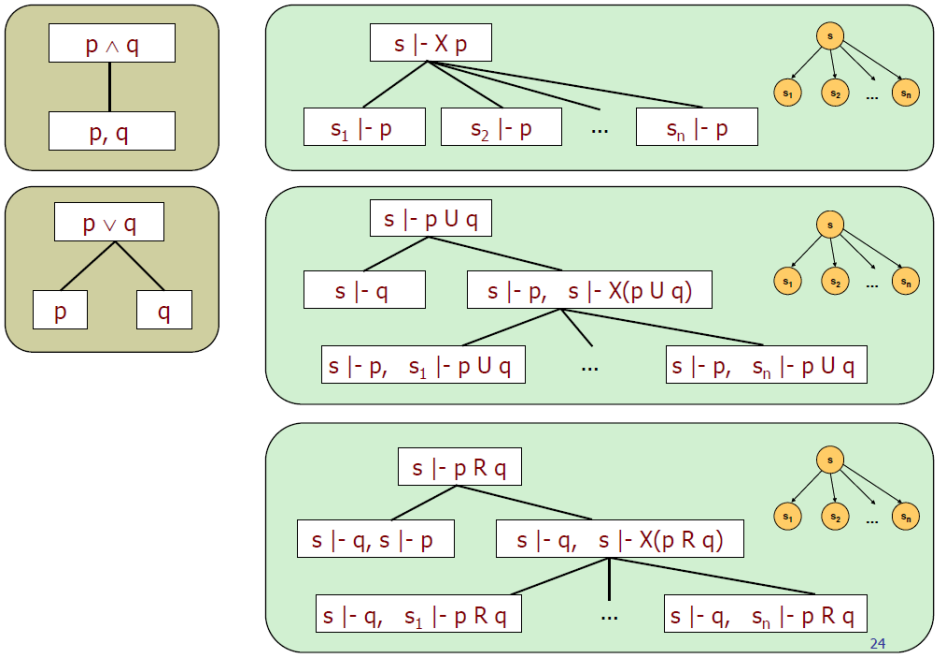
4. Modellellenőrzés tabló módszerrel: A tabló módszer elve és a tabló alapú modellellenőrzés PLTL esetén.

Tabló módszer: annak a megvizsgálására, hogyan tehető igazzá egy adott kifejezés (Boole-függvény). A kifejezést *fa struktúrába* bontjuk fel, csomópontjai a kifejezések, amelyeket igazzá akarunk tenni, azokat felbontási szabályokkal bontjuk szét, a kifejezésben szereplő operátorok jelentése alapján. Felbontás előtt negálásokat *negált normál formára* kell hozni (pl de Morgan azonosságokkal), vagyis csak változók előtt legyen negált, összetett kifejezések előtt ne.

Terminálás: nem maradt operátor, csak változók listája (ponált vagy negált alakban). Ellentmondásos az ág, ha vannak ugyanazok a változók negált és ponált alakban is, akkor nem lehet úgy behelyettesíteni, hogy igaz legyen a kifejezés. A fa sikeres ágai mutatják meg, hogy tehető igazzá a kifejezés.

Használat PLTL-hez: ellenpéldát keresünk a kifejezésre: kifejezés negáltjának a tablóját írjuk fel! Fa sikeres ágai ellenpéldát adnak. Ha nincs ellentmondásos ág, az eredeti kifejezés igaz! Temporális operátorokhoz új felbontási szabályok kellenek. A modell ismeretében bonthatóak fel.

Jelölés: **s|- p**, az **s** állapotban vizsgáljuk, hogy **p** igaz e. Atomi P kifejezésnél s|- P igaz, ha P∈L(s)
p U q negálásához R operátor: ¬(p U q) = (¬p) R (¬q), p R q = q ^ (p ˅ (p R q))



5. Modellellenőrzés szemantika alapon: A szemantikán alapuló modellellenőrzés alkalmazása CTL esetén.

Globális modellellenőrzés p kifejezésre. Modell összes olyan állapotát felcímkézzük p-vel, ahol az igaz. A modellen akkor igaz p, ha a kezdőállapotán szerepel a p címke.
Címkézés eredeti kifejezés részkifejezéseivel: atomi kijelentések, majd belőlük alkotott összetett részkifejezések p-ben, végül az egész p kifejezés.
Belülről kifelé haladva címkézzük a Kripke-struktúra állapotait. Ha a p és q címkék már kikerültek, felírható az állapotokra, hol teljesülnek az összetett kifejezések:

- ¬p: p nem eleme L(s)-nek
- p^q: p és q is eleme L(s)-nek
- EX p: ha van rákövetkező állapot, ahol p eleme L(s)-nek
- AX p: ha minden rákövetkező állapotban p eleme L(s)-nek
- E(p U q): ha s címkézett q-val, vagy p-vel és legalább egy következője címkézett E(p U q)-val. Visszafelé iterálunk, ahol van q, rátehető az E(p U q), aztán megelőző állapotain ahol van p, rátehető oda is.
- A(p U q): ha s címképzett p-vel,és minden következője címkézett A(p U q)-val. Iteráció visszafelé, ahol van q, rátehető A(p U q), megelőző állapotaiban, ahol van p, és az állapot minden rákövetkezőjén van A(p U q), arra is rátehető.
- AF p: ha s címkézett p-vel, vagy ha minden rákövetkező már címkézett AF p-vel. Iteráció visszafelé, ahol van p, rákerülhet AF p, megelőzőinél pedig szintén rákerülhet, ahol minden rákövetkezőnél van AF p. Vagyis visszafelé keressük azokat az állapotokat, amik minden útvonalon p-vel címkézett állapotokba vezetnek!

Iteráció halmazműveletekkel: Megelőző állapotok címkézett Z állapothalmaznál:

$pre_E(Z) = \{ s \in S \mid \exists s', (s, s') \in R \text{ és } s' \in Z \}$ (legalább egy rákövetkezője címkézett Z-ben!)

$pre_A(Z) = \{ s \in S \mid \forall s', \text{ahol } (s, s') \in R, s' \in Z \}$ (minden rákövetkezője címkézett Z-ben)

Z_0 kezdeti felcímképzett állapotok, majd iterálva a művelet alapján, meglévők uniójához hozzávéve az újakat. Iteráció véget ér, ha $Z_{i+1}=Z_i$

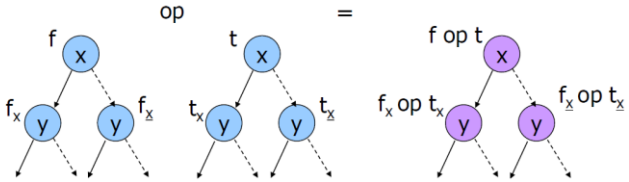
6. Bináris döntési diagramok: Az ROBDD származtatása, felépítése és tulajdonságai. Logikai függvény átírása ROBDD alakba. Logikai műveletek végrehajtása az operandusok ROBBD reprezentációjával dolgozva.

Bináris döntési diagrammok:

- Boole függvények bináris döntési fa alakban: x változó behelyettesítése, két ágra bontjuk: 0 vagy 1. Ez a tesztváltozó, értékének vizsgálata a teszt. A függvény egy változója ezzel eltűnt. Tovább folytatjuk, míg van változó.
- Azonos részfákat összevonva => **BDD** *bináris döntési diagram*
- Ha minden felbontáskor azonos sorrendben vesszük fel a teszt változókat => **OBDD** *rendezett bináris döntési diagram*
- Ha a szükségtelen csomópontokat töröljük => **ROBDD** *redukált rendezett bináris döntési diagram*

ROBDD: irányított, aciklikus gráf, egy gyökérrel és két levéllel (0, 1), minden csomópontban egy teszt változóval. Minden csomópontból 2 él indul ki, azt jelölve, hogy a változóba 0 (szaggatott vonal) vagy 1 (folytonos vonal) értéket helyettesítünk. Adott függvény esetén két, azonos változósorrendezésű ROBDD izomorf.
Mérete függvényenként változó, egészen kompakttól (páros paritás) akár exponenciálisan nagy (XOR) is lehet. Fontos a változók sorrendezése!

Két azonos változókkal rendelkező függvényeken végzett Boole operátorokat végrehajthatunk közvetlenül ROBDD-ken is, ha azokban a változók sorrendezése megegyezik! 1 ágat 1 ággal, 0 ágat 0 ággal kell össze vonni. (Ha redukálások miatt egyik ROBDD-ben hiányzik egy változó, a következő szinten lévő csomópont értékét vesszük az egyesítéskor.)



Előnyei: kanonikus alak, függvények ekvivalenciája jól vizsgálható. Algoritmusok hatékonyan gyorsíthatók általa. Változók sorrendjétől függően tárigény csökkenés érhető el.

7. A szimbolikus modellellenőrzés alapötlete, a karakterisztikus függvény fogalma. A szemantika alapú modellellenőrzés megvalósítása ROBDD felhasználásával.

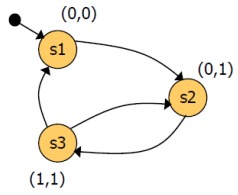
CTL modellellenőrzés eddig *szemantika alapú módszerrel*: állapotok iteratív címkézésével.

Szimbolikus technika: Állapothalmazok helyett logikai függvények kezelése. Alapötlet: *állapotok „kódolása”* n bites bitvektorral (n az állapotok számának 2 alapú logaritmusának felső egész része).

Állapothalmazt **C: {0,1}^{n}->{0,1}** *karakterisztikus függvény* írja le: karakterisztikus függvény igaz egy bitvektorra, ha a bitvektor által kódolt állapot az adott állapothalmazban van. **$C_s(x_1,x_2,\dots,x_n)$** karakterisztikus függvényt ^ operátorral konstruáljuk úgy, hogy egy **$(u_1,u_2,\dots,u_n)$** bitvektor esetén *ponált* az **x_i**, ha **u_i**=1, egyébként *negált*.

Az **S** állapothalmaz egy **Y** részhalmazára úgy konstruáljuk a függvényt, hogy minden állapotra felírt karakterisztikus függvényt egyesítjük ᵛ operátorral.

Állapotátmenet karakterisztikus függvénye dupla annyi változós: a *kezdőállapot* és *célállapot* kódolt bitvektorát kapja paraméterül. A két állapot karakterisztikus függvényének ^ operátorral való egyesítése adja a karakterisztikus függvényt.



Változók: x, y

Állapotok karakterisztikus függvényei:

s1 állapot:

$$C_{s1}(x,y) = (\neg x \wedge \neg y)$$

s2 állapot:

$$C_{s2}(x,y) = (\neg x \wedge y)$$

s3 állapot:

$$C_{s3}(x,y) = (x \wedge y)$$

$(s1,s2) \in R$ állapotátmenet:

$$C_{(s1,s2)} = (\neg x \wedge \neg y) \wedge (\neg x' \wedge y')$$

Állapotátmeneti reláció: Összes átmenet

$$R(x,y,x',y') = (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee (\neg x \wedge y \wedge x' \wedge y') \vee (x \wedge y \wedge \neg x' \wedge y') \vee (x \wedge y \wedge \neg x' \wedge \neg y')$$

Állapothalmaz karakterisztikus függvénye:

{s1,s2} állapothalmaz:

$$C_{\{s1,s2\}} = C_{s1} \vee C_{s2} = (\neg x \wedge \neg y) \vee (\neg x \wedge y)$$

Szemantika alapú modellellenőrzés: állapothalmaz műveletek visszavezethetőek logikai függvényeken végzett műveletekre. Kódolás után ROBDD-ken végzett műveletekkel elvégezhető a modellellenőrzés.

- halmazok *uniója* = függvények ᵛ kapcsolata
- *metszete* = ^ kapcsolata
- $pre_E(Z)$ is előállítható karakterisztikus függvényként, *egzisztenciális absztrakcióval*: Z állapothalmazt leíró C_Z függvény, R átmeneteket leíró C_R függvény, $pre_E(Z)$-t leíró karakterisztikus függvény: $C_{pre(Z)} = \exists_{x1',x2',\dots,xn'} C_R \wedge C_Z'$, ahol $\exists_x C = C[1/x] \vee C[0/x]$

8. Korlátos modellellenőrzés: A korlátos modellellenőrzés alapötlete. A modellellenőrzés algoritmusa. A szoftver modellellenőrzés jellegzetes problémái.

Adott logikai függvényhez a *SAT-megoldók* képesek olyan változó-behelyettesítést keresni, mellyel a függvény értéke igaz. Ha a problémát le tudjuk képezni egy logikai függvényre, a *modell és temporális logika* segítségével, SAT-megoldót használhatunk a modellellenőrzésre.

Követelmények általában *invariáns követelmények*, minden állapotre előírt tulajdonságok. Követelmény teljesülésekor a megoldó nem talál behelyettesítést. Ellenkező esetben a talált megoldás ellenpéldát ad a teljesülésre, amit használhatunk a hibakereséshez. Invariáns tulajdonságok esetén jól használható módszer.

Korlátos modellellenőrzés: Útvonalak hosszát korlátozva, iteratívan végezzük a modellellenőrzést.
Tesztgenerálásra jól használható.
Kezdőállapotot leírja **I(s)** karakterisztikus függvény.
Következő állapotokat $\mathbf{C_R(s^0,s^1)}$ *állapotátmeneti karakterisztikus függvény* segítségével kapjuk meg, ugyanígy tovább $C_R(s^1,s^2)$.
Követelmény invariánsok: minden állapotban teljesülnie kell a **p(s)** predikátumnak (címkézett állapotok halmazának karakterisztikus függvénye.
Ellenpéldát találunk, ha valahol nem teljesül p(s).
path(): *k hosszú útvonalak* (k+1)*n változós karakterisztikus függvénye, út során szereplő állapotátmenetek karakterisztikus függvényei egyesítése ^ operátorral.
Algoritmus:

- iteráció útvonalak hosszára i=0,1,2…
- ciklusmentes utakat vizsgálunk!
- A SAT-megoldó a következő függvényre keres megoldást (esetünkben ellenpéldát): $I(s^0)$ ^ $path(s^0,s^1,\dots,s^i)$ ^ $\neg p(s^i)$
- Iteráció megáll, ha kezdőállapotból nem létezik i hosszú ciklusmentes út, vagy egyáltalán nem létezik i hosszú ciklusmentes út, ami olyan állapotba vinni, ahol nem igaz p(s). Ekkor p(s) mindenhol igaz.
- Az, hogy nem találtunk ellenpéldát, nem ad teljes bizonyosságot: lehet, hogy több iterációval találnánk.

Az algoritmus finomítható azzal, ha nem 0-ról, hanem egy k számtól kezdjük az iterálást. Ez nem biztosítja a minimális hosszúságú ellenpéldát, lehet, hogy túllövünk a célon (heurisztika segíthet). További megkötések szükségesek a SAT megoldó terminálásához: a kezdőállapotból vezető útvonalakon ne legyenek kezdőállapotok, és az ellenpéldához vezető útvonalakon ne legyenek további „rossz” állapotok.

Szoftver modellellenőrzés problémák: *ciklusok bejárása* új állapotokat eredményez, állapotváltozók módosulnak. *Teljes kihajtogatás* esetén szisztematikusan bejárjuk előre az összes lehetséges ágat (F-SOFT eszköz), másik megoldás azonban a *korlátozott ciklusbejárás* (CMBC, SATURN eszközök), ahol minden ciklusra lefutási korlátot adunk, csak annyi elágazásig nézzük meg.
Általános modellellenőrzés hiányosságok a skálázhatósággal kapcsolatos problémák, komplex adatstruktúrák állapotterének bejárásának erőforrás igényessége, nehéz az eredményeket általánosítani (protokoll 2 résztvevőre jó, de N-re??), és a követelmények formalizálása sem egyszerű.

9. Az állapottérképek elemkészlete: Állapotok és finomításuk, emlékező állapotok, az állapotkonfiguráció fogalma. Állapotátmenetek és fajtáik. Akciók és fajtáik.

Állapottérkép: állapot alapú, eseményvezérelt rendszerek modellezésére, állapotgép viselkedésének leírására. Reaktív viselkedés, külső események hatására történő állapotváltást ír le. Szokásos használatuk pl. beágyazott rendszerek esetén a bejövő események feldolgozásának modellezésére, forráskód generálására, modellellenőrzésre, tesztek generálására, futási idejű verifikációra (monitorkód generálás).

Állapot: alapszintű modellelem, adott feltételek teljesülését jelenti, állapotváltozóknak meghatározott értékük van. Belépési akciók, kilépési akciók, belső aktivitások tartozhatnak hozzá.
Állapot finomítás: az állapotok egymásba ágyazódhatnak, állapothierarchiát hova létre. A több állapotot tartalmazó „szuperállapot" foglalja magába a közös tulajdonságokat a többi belső állapotban.

- *Egyszerű állapot*nak nincs finomítása
- *OR jellegű finomítás*sal alárendelt állapotokat írunk le, ezek közül egyszerre egy lehet aktív
- AND jellegű finomítással konkurens régiókat hozunk létre, egyidejűleg minden régióban kell legyen aktív állapot.

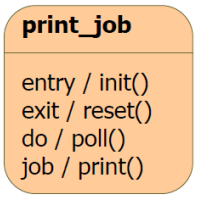
Állapot konfiguráció: leírja a jelenleg aktív állapotokat, aktív összetett állapot esetén az összes aktív állapot megadásával, elosztott végrehajtás esetén mindkét szálon lévő aktív állapotokkal.
Emlékező állapot: legutolsó aktív állapotkonfigurációt „tárolja", lehetőséget ad egy korábbi aktív állapotkonfigurációra való visszatérésre, pl. közbenső esemény feldolgozása után. A kimenő átmenet jelöli meg, ha nem volt még ott aktív állapot, milyen konfigurációba menjünk, vagyis mi az alapértelmezett állapot.

- *Egyszerű emlékező* állapot csak adott finomítási szinten emlékezik,
- *Mélyen emlékező* állapot mélyebb finomítási szinteket is megjegyzi.

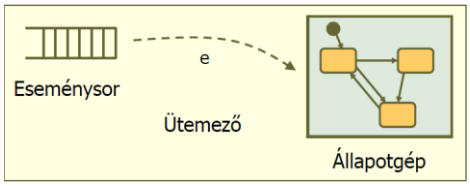
Állapotátmenet: állapot változása, *trigger esemény* válthatja ki, ami egy aszinkron történés, aminek paraméterei is lehetnek, de *trigger nélküli* állapotátmenet is lehetséges, azok „önmaguktól" tüzelnek. *Time-out* is lehet egy trigger, az átmenet tüzel, ha az állapotgép a hozzá tartozó kiindulási állapotban tartózkodik az adott időintervallumban.
Őrfeltétel rendelhető hozzá, az állapotátmenet csak akkor történhet meg, ha az őrfeltétel igaz. Állapotváltozók és esemény paraméterek szerepelhetnek benne.
Akció is rendelhető hozzá, az állapotátmenetkor a hozzá rendelt tevékenység is végrehajtódik.
Jelölés: **trigger [guard] / action**
Összetett állapotátmenet:

- *Szétváló*: konkurens régiókba lévő állapotokba való együttes belépés
- *Egyesülő*: konkurens régiókban lévő állapotokból való együttes kilépés
- *Elágazó*: több, őrfeltételtől függő átmenet egyszerűsített jelölése.

Akciók: Állapotok belépési, belső és kilépési akciói, és átmenetekkel járó akciók.
Állapotátmenetkor tüzelési sorrend: konkurens régiókban párhuzamosan kilépési akciók + átmenet akciója + belépési akciók.



10. Állapottérképek informális szemantikája: Eseménykezelés fogalma és menete. A „run-to-completion” elv. Állapotátmenetek engedélyezettségének kritériumai, a konfliktus fogalma és feloldása, az állapotátmenetek tüzelése, az akciók sorrendezése.



Állapottérkép szemantikája: Az eseményfeldolgozás a szemantika lényege, ez alapján lehet programkód alakjában megvalósítani az állapotgépet (forráskód generálás).
Egy *állapotgép* írja le az állapottérkép viselkedését, mit tesz az állapotgép egy esemény hatására.
A „futtató rendszert” egy *eseménysor* és egy *ütemező* biztosítja, amik az állapotgép szempontjából külső elemek.
Az ütemező által adott események hatására az *állapotátmenetek tüzelnek* (akár több konkurens állapotátmenet is), és az *állapotkonfiguráció változik*.
Az ütemező akkor küld újabb eseményt, ha az előző feloldogzása teljesen megtörtént. *Egyenként dolgozzuk fel az eseményeket*.
Run to completion elv: események teljes felolgozását végezzük, minden lehetséges átmenet tüzel, átmenetek maximális halmaza! Konfliktusok lehetnek, azokat fel kell oldani.
Eseményfeldolgozás lépései:

1. Engedélyezett átmenetek megállapítása
2. Tüzelő átmenetek kiválasztása, konfliktusfeloldás
3. Átmenetek tüzelése, akciók végrehajtása
4. Új konfigurációba lépéssel járó teendők, aktív állapotok meghatározása

Engedélyezett állapotátmenet: ha a kiindulási (forrás-) állapot aktív, a kiválasztott esemény az átmenet triggere, és az őrfeltételek teljesülnek.

- 1 engedélyezett: tüzelhet
- 0 engedélyezett: ha halasztott esemény, tároljuk, és új eseményt kérünk az ütemezőtől. Egyébként hatás nélkül eldobjuk.
- Több engedélyezett: tüzelő átmenetek kiválasztása szükséges, ha kell, konfliktus feloldással.

Konfliktusfeloldás: Két átmenet konfliktusban van, ha mindketten ugyanazt az állapotot hagyják el, vagyis az elhagyott állapothalmazok metszete nem üres. Feloldás szükséges, hogy eldöntsük, melyik tüzeljen.

- *Prioritás alapján*: egy átmenet *prioritása* nagyobb, ha az átmenet kiindulási állapota az állapothierarchiában (állapotfinomításban) alacsonyabb szintű
- *Véletlenszerűen*: ha azonos prioritású átmenetek tudnak csak tüzelni

Állapotátmenet tüzelése: kiválasztott átmenetek (konfliktusfeloldás után) véletlenszerű sorrendben tüzelnek, akcióik közötti sorrend is véletlenszerű. Egy átmenet tüzelése során akciók sorrendje:

1. Kiindulási állapotok elhagyása: alacsonyabb hierarchia szintről indulva kilépési akciók (exit) végrehajtása
2. Átmenet akcióinak végrehajtása
3. Célállapotokba való belépés: magasabb hierarchia szintről befelé haladva az állapotok belépési (entry) akciók végrehajtása.

Új konfigurációba való belépés:

- Egyszerű állapot: új konfiguráció része lesz, és ős állapotai is aktívak lesznek. Aktív ős állapot minden konkurens régiójában lesz aktív állapot, melyet a kezdőállapot jelöl ki.
- Összetett (OR finomítású) állapot: finomításban a kezdőállapot jelöli ki az aktív állapotot.
- Konkurens régiókkal rendelkező (AND finomítású) összetett állapot: kezdőállapotok jelölik ki az aktív állapotokat minden régióban.
- Nem stabil állapotból azonnal továbblépünk.

11. Petri hálók alapfogalmai: Formális definíció. Engedélyezettség és tüzelés fogalma. Kiterjesztések (tiltó élek, prioritás, kapacitáskorlát). A kiegészítő helytranszformáció. Az egyszerű és a kiterjesztett Petri hálók kifejezőereje.

Használható konkurens, aszinkron, elosztott, párhuzamos, nemdeterminisztikus rendszerek modellezésére. Egyidejűleg biztosítja a grafikus reprezentációt és a matematikai formalizmuzt, áttekinthető, precíz és egyértelmű. Struktúrával fejezi ki a vezérlési struktúrát és adatstruktúrát. További előnye, hogy könnyen kiterjeszthető, és más ábrázolásmódok is leképezhetők Petri hálóvá.

Petri háló (PN): $PN = \langle P, T, E, W, M_0 \rangle$

- Helyek: $P = \{p_1, p_2, \ldots, p_\pi\}$
- Tranzíciók (tüzelések): $T = \{t_1, t_2, \ldots, t_\tau\}$, $P \cap T = \varnothing$
- Élek: $E \subseteq (P \times T) \cup (T \times P)$
- Súlyfüggvény: $W : E \to \mathbf{N}^+$
- Kezdőállapot: $M_0 : P \to \mathbf{N}$

PN struktúra: $N = \langle P, T, E, W \rangle$

PN adott kezdőállapottal: $PN = \langle N, M_0 \rangle$

- Helyek és tranzíciók bemeneti és kimeneti elemei:
  - $t \in T$ bemeneti helyei: $\bullet t = \{p \mid (p,t) \in E\}$
  - $t \in T$ kimeneti helyei: $t \bullet = \{p \mid (t,p) \in E\}$
  - $p \in P$ bemeneti tranzíciói: $\bullet p = \{t \mid (t,p) \in E\}$
  - $p \in P$ kimeneti tranzíciói: $p \bullet = \{t \mid (p,t) \in E\}$
- Csomópontok $P' \subseteq P$ és tranzíciók $T' \subseteq T$ részhalmazára:

$$\bullet P' = \bigcup_{p \in P'} \bullet p \qquad \bullet T' = \bigcup_{t \in T'} \bullet t$$

$$P' \bullet = \bigcup_{p \in P'} p \bullet \qquad T' \bullet = \bigcup_{t \in T'} t \bullet$$

Alapfogalmak: Páros gráf két típusú csomóponttal, **hely** (p) és **tranzíció** (t), melyek között irányított élek mennek: csak hely->tranzíció és tranzíció->hely élek.
Tokenek jelölik az állapotokat, amelyek tranzíciói tüzelhetnek, ha a feltételek megfelelőek. A helyek kapacitáskorlátja végtelen. Háló állapota leírható egy *állapotvektor*ral, ahol x_i értéke az i. helyen lévő tokenek száma.
Élek **élsúlyokkal** rendelkezhetnek. Tranzíció bemenő élén lévő súly jelöli, hány token szükséges a tüzeléshez a bementi helyen, és a kimenő élen lévő súly jelöli, a kimeneti helyen hány token keletkezik.

Engedélyezett tranzíció: Tranzíció akkor tüzelhet, ha engedélyezett: minden bementi helyen legalább annyi token van, amennyi a tranzícióba vezető él súlya.

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

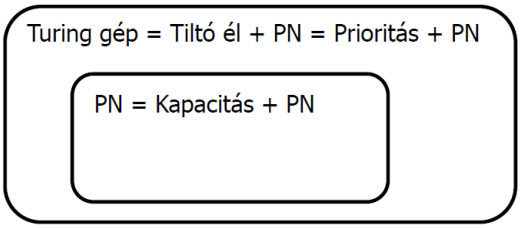
Tranzíció tüzelése: Adott egy kezdeti tokeneloszlás. Engedélyezett tranzíció „tetszés szerint" tüzelhet („fire at will"). Tüzeléskor a tokent elvesszük a kimeneti helyről, és minden kimeneti helyre tokent helyezünk (nem mozgatjuk, több és kevesebb is keletkezhet, eltűnhet). A tüzelés után új tokeneloszlás jön létre.
Implicit időfogalom: nincs időzítés, a tüzelés a [0, ∞) időintervallumban bármikor megtörténhet. Egyszerre csak egy tranzíció tüzelhet. Nemdeterminisztikus működés, ha több tranzíció is engedélyezett, véletlenszerűen kell választani egyet.

Kiterjesztése: cél a modellezési erő növelése, működés nemdeterminizmusának korlátozása

- *Kapacitáskorlát helyekhez*: minden p helyhez **K(p) kapacitás**: maximálisan annyi token lehet ott (*véges kapacitású Petri-háló*). Tüzelhet egy tranzakció, ha engedélyezett, és tüzelés után is minden helyen teljesülnek a kapacitás korlátok.
  Végtelen kapacitású Petri-hálóval is megoldható a kapacitáskorlát: legyen p hely kapacitáskorláta K(p), kezdeti tokenjeinek száma x. Vegyünk fel egy új p' adminisztrációs helyet, ahova a ki nem használt kapacitásnak megfelelő számú tokent helyezünk! Minden tranzícióhoz, aminek kimenő helye a p, húzzunk bemenő élet ebből a p' helyről, és minden tranzícióhoz, aminek bemenő helye a p, húzzunk élt a p' helybe. A két háló tüzelési szekvenciái azonosak.

- *Tiltó élek*: „negált” tüzelési felvétel: a tüzelés a feltétel megléte esetén nem hajtódhat végre! Tranzíció nem tüzelhet, ha bármely tiltó él bemenő helyén legalább annyi token van, amennyi az él súlya. Jelölés: kör az él végén. Kifejező ereje nagyobb, mint a sima Petri-hálóknak (Turing gép szintje)! (de az analízist bonyolultabbá teszik)
- *Prioritás tranzakciókhoz*: egyszerre engedélyezett tranzakciók során nemdeterminizmus helyett a tranzakciókhoz rendelt prioritás alapján dönt, melyik tüzelhet. Tranzakció tüzelhet, ha engedélyezett, és nincs nála magasabb prioritású engedélyezett tranzakció. (Azonos prioritás esetén itt is véletlenszerű.)

Kifejezőerő: tiltó él és prioritás képes „zero testingre”. „Zero testing” képesség lehetővé teszi, hogy minden Turing gép szimulálható Petri hálóval. A kapacitáskorlát csak szintaktikus konstrukció. Vannak olyan rendszerek, amik nem modellezhetők kiterjesztések nélküli Petri hálóval (pl. adott helyen k számú token van e).



12. Petri hálók szimulációja: Token játékok. Egyszerű, prioritásos és időzített Petri hálók szimulációja (algoritmusok). Tevékenység modellezés, erőforrás modellezés Petri hálókban.

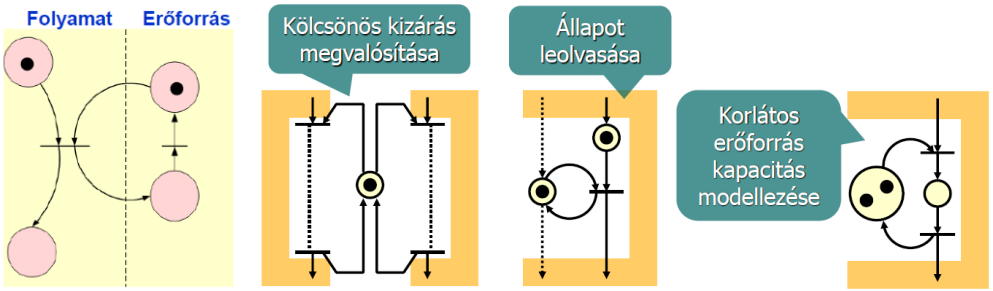
Szimuláció: a rendszer lehetséges trajektóriáinak vizsgálata. Állapot = tokeneloszlás, állapotváltás = tranzíció tüzelés, trajektóriák = bejárható állapotsorozatok tüzelési szekvenciák hatására.
Interaktív szimuláció: felhasználó választ a tüzelhető tranzíciók közül.
Automatikus szimuláció: álvéletlen generálás alapján, felhasználó csak a tüzelések számát állítja be. Statisztika gyűjtésére hasznos: tüzelések száma, aránya, helyek átlagos tokenszáma.
A tokenjáték szabályai szerint egy tüzelésre kész tranzíció a döntést követően atomi lépésben azonnal lejátszódik.

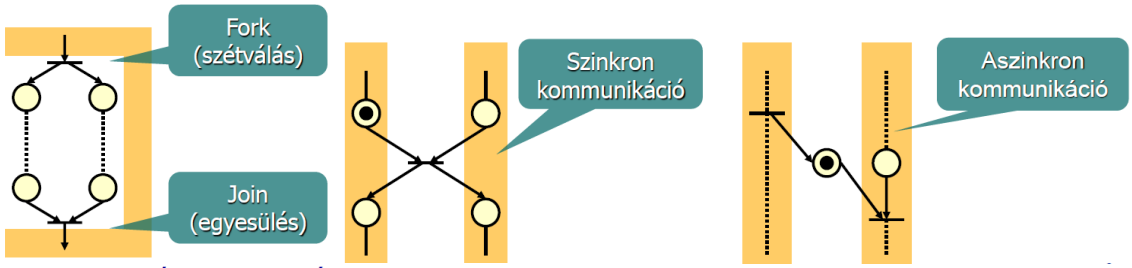
Algoritmusok:

- *Egyszerű*: tüzelhető tranzíciók listájának összeállítása, majd while ciklusban addig iterálunk, míg van tüzelhető tranzíció: nemdeterminisztikus választás, tüzelés, tüzelhető tranzíciók listájának összeállítása.
- *Hatékonyabb*: Ahelyett, hogy mindig végig iterálnánk az összes tranzíción, ellenőrizve, engedélyezett-e, elég ezt először megtenni, majd minden tüzelés után azokat a tranzakciókat ellenőrizni, melyre a tüzelés hatással lehetett: nézzük a tranzíció bemenő helyeinek kimenő tranzícióit. Ha tüzelés után letiltódtak, vegyük ki őket a listából. Ugyanígy nézzük a tranzíció kimenő helyeinek kimenő tranzícióit, ha engedélyezetté váltak, vegyük fel őket a listába.
- *Prioritásos*: nem halmazba gyűjtjük a tüzelhető tranzíciókat, hanem halmazok vektorába, ahol az L[i] halmaz tartalmazza az i prioritású tüzelhető tranzíciókat. Tüzeléskor a legmagasabb prioritású nem üres L[i] halmazból választunk nemdeterminisztikusan.
- *Időzített*: Az Időzített Petri hálókban léteznek olyan tranzíciók, melyek csupán egy külső időhöz kötötten (pl. órajel) tüzelhetnek. Az időzítés bevezetésével a tokenjáték „lejátszhatóvá" válik, nemcsupán léptethetővé, ezért a tranzícióknak egy további új tulajdonsága is megjelenik, mégpedig a késleltetés. Ezáltal „lejátszási" módban az időzítéshez nem kötött tranzíciók nem azonnal, hanem a beállított várakoztatást követően tüzelnek csupán. Általános eloszlásfüggvény adható a tranzíciók tüzelési idejének (késleltetésének) sorsolásához. A késleltetések újrasorsolásának szemantikája egy-egy új tokeneloszlás esetén nem triviális!

Modellépítés folyamata:

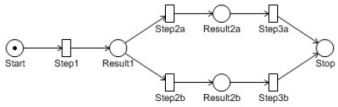
1. *Tevékenységek* (folyamat) modellje: sorrendezés, erőforrás használat. Üzenetváltás még mindegy.
2. *Erőforrások* modellje: foglalt/szabad, üzenetek tárolója.
3. *Interakciók* szerinti integrálás: változások hatása, átmenetek összevonása a folyamat és erőforrás modellekben (pl. „foglalás" összevonása a „szabad-> foglalt" átmenettel). Tevékenységek feltételei, élek bekötése az erőforrás állapotokból a tevékenységekhez (pl. „hibamentes" állapot szükséges egy tevékenység indításához).



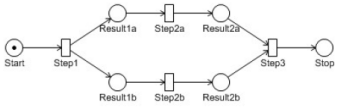


- Szekvenciális feldolgozás:

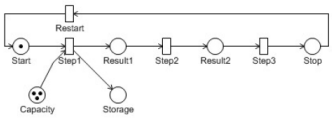
- Alternatív feldolgozás:



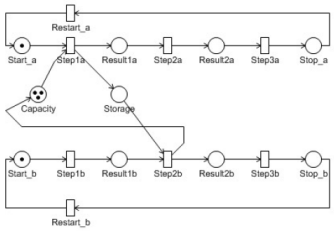
- Párhuzamos feldolgozás:



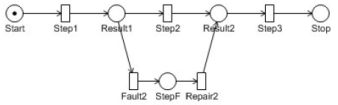
- Véges kapacitású tároló (betelik):



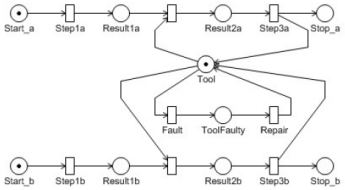
- Termelő és fogyasztó folyamat (interakció a tárolón):



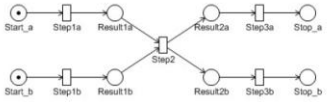
- Hibás tevékenység a folyamat során:



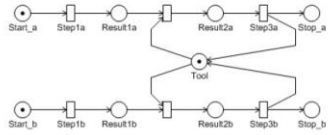
- Erőforrás (itt: megosztott erőforrás) meghibásodása a feldolgozás során:



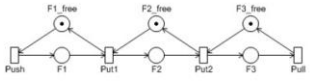
- Szinkronizálás (randevú):



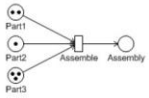
- Megosztott erőforrás (gép, eszköz, munkaerő):



- FIFO tároló:



- Alkatrészek összeszerelése:



13. Petri hálók dinamikus tulajdonságai: Az elérhetőség fogalma. Korlátosság, élőség, megfordíthatóság, visszatérő állapot, fedhetőség, perzisztencia, korlátozott és globális fairség tulajdonságok és jellegzetes felhasználásuk.

Elérhetőség: az állapottér szolgál a kezdőállapotfüggő viselkedés leírására. Elérhetőségi gráfként ábrázolhatjuk, a helyeket tokeneloszlásukkal leírva, a tranzakciókat pedig állapotátmenetekként feltüntetve. Egy M_n állapot elérhető az M_0 kiinduló állapotból, ha van olyan σ tüzelési szekvencia, ami M_n állapotba visz.

$$\exists \vec{\sigma} : M_0 \left[\vec{\sigma} > M_n \right.$$

*Elérhetőségi analízis*kor arra kereshetjük a választ, mely állapotok elérhetőek, illetve milyen tüzelési sorozatok hajthatóak végre (amik létező állapotba visznek).
Elérhetőségi problémák eldönthetőek, de általában exponenciális komplexitásúak. Rész-tokeneloszlási probléma: helyek egy P' halmazára korlátozzuk a kérdést, elérhető-e egy állapot az adott helyekre megadott tokeneloszlással?

Korlátosság: k-korlátosságnak nevezzük, ha bármely állapotban minden helyen maximum k token lehet. A „végesség" kifejezésére használjuk, az állapottér véges. Speciális eset a k=1, ez a *Biztos Petri háló*. Megválaszolható gyakori kérdések: felgyűlnek-e a rendszerben feladatok, megvalósul-e az üzenetek rendszeres feldolgozása?

Élőség: Egy Petri háló Lx-élő (x=1,2,3,4), ha minden tranzíciója Lx-élő. Holtpontmentes, ha minden állapotban legalább egy tranzíció tüzelhető. Petri háló élő, ha L4-élő, és holtpontmentes (bejárási úttól függetlenül, garantáltan!)! (Tehát élő => holtpontmentes, de nem igaz, hogy holtpontmentes => élő)

- L0-élő (halott): t SOHASEM tüzelhet
- L1-élő (gyengén): t LEGALÁBB EGYSZER tüzelhet
- L2-élő (gyengén): t LEGALÁBB k-SZOR (bármely k>1 egészre)
- L3-élő (gyengén): t VÉGTELEN SOKSZOR tüzelhet
- L4-élő (erősen): t BÁRMELY ÁLLAPOTBAN LEGALÁBB EGYSZER tüzelhet

Megfordíthatóság: Az M_0 kezdőállapot elérhető bármely őt követő állapotból. Gyakori példák rá a „reset" jellel kezdőállapotba vihető rendszer, ha a biztonságos kezdőállapot mindenhonnan elérhető, vagy a kezdőállapotot is magába foglaló ciklikus működésű hálózatok.

Visszatérő állapot: A kezdőállapotból elérhető M állapot elérhető bármely őt követő állapotból. Gyakori példák rá az inicializáló szekvencia után egy ciklikus működés adott állapotokon keresztül (ők a visszatérő állapotok), vagy az inicializálás után bárhonnan elérhető biztonságos állapot (ő a visszatérő állapot)

Fedhetőség: Létrejön-e korábbi állapotot megában foglaló állapot? Minden helyen legalább annyi (gyenge fedhetőség a nagyobb egyenlő, erős a nagyobb) token van, mint egy megelőző állapotban.
Kapcsolat élőséggel: ha egy tokeneloszlásban engedélyezett a t tranzíció, t akkor és csak akkor NEM L1-élő, ha a tokeneloszlás nem fedhető le! Tehát a tokeneloszlás fedhetősége garantálja t L1-élő voltát (vagyis tüzelhet).
Fedési gráfot rajzolhatunk hozzá. Leolvasható róla, ha a háló korlátos, biztonságos, van halott tranzíciója

Perzisztencia: Egy tranzíció perzisztens, ha engedélyezetté válása után tüzelésig az is marad, tehát nincs olyan engedélyezett tranzíció, amely tüzelése letiltja őt (elveszi tőle a tokent). Egy Petri háló perzisztens, ha bármely két tranzíciója az összes lehetséges tüzelési szekvenciában perzisztens. Gyakori példák az alkalmazásra annak vizsgálata, a párhuzamos működések befolyásolják-e egymást, és a rendszerbeli funkcionalitás dekompozíció után megmaradt-e?

Fairség: Gyakori példák az alkalmazására annak a vizsgálata, hogy a párhuzamos folyamatok nem tartják-e fel egymást, valamennyi folyamat végbemegy-e előbb-utóbb, egy kérés kiszolgálása megtörténik-e előbb-utóbb?

- *Korlátozott fairség* (B-fair): tüzelési szekvencia korlátozottan fair, ha bármely tranzíció maximum korlátos sokszor tüzelhet anélkül, hogy egy másik engedélyezett tranzíció tüzelne. Nem veheti el folyamatosan a tüzelési lehetőséget más elől. Petri háló korlátozottan fair, ha az összes lehetséges tüzelési szekvenciája korlátozottan fair!
- Globális fairség: ha a tüzelési szekvencia véges, vagy az összes tranzíció végtelen sokszor tüzelhet benne. Petri háló globálisan fair, ha az összes lehetséges tüzelési szekvenciája globálisan fair.

14. Petri hálók elérhetőségi analízise: Az elérhetőségi és fedési gráf generálása (algoritmusok). Az elérhetőségi analízis előnyei és hátrányai. Az elérhetőségi probléma egyszerűsítése (struktúra redukció).

Elérhetőség: az állapottér szolgál a kezdőállapotfüggő viselkedés leírására. Elérhetőségi gráfként ábrázolhatjuk, a helyeket tokeneloszlásukkal leírva, a tranzakciókat pedig állapotátmenetekként feltüntetve. Egy M_n állapot elérhető az M_0 kiinduló állapotból, ha van olyan σ tüzelési szekvencia, ami M_n állapotba visz. *Elérhetőségi analízis*kor arra kereshetjük a választ, egy M_0 kiinduló állapotból mely állapotok elérhetőek, illetve milyen tüzelési sorozatok hajthatóak végre (amik létező állapotba visznek). *Elérhetőségi problémák* eldönthetőek, de általában exponenciális komplexitásúak. Rész-tokeneloszlási probléma: helyek egy P' halmazára korlátozzuk a kérdést, elérhető-e egy állapot az adott helyekre megadott tokeneloszlással?

$$\exists \vec{\sigma} : M_0 [\vec{\sigma} > M_n$$

$$R(N, M_0) = \{M \mid \exists \vec{\sigma} : M_0 [\vec{\sigma} > M\}$$

$$L(N, M_0) = \{\vec{\sigma} \mid \exists M : M_0 [\vec{\sigma} > M\}$$

Elérhetőségi gráf: M_0 kezdőállapotból induló állapotgráf. Csomópontok az állapotok, címkézésük a tokeneloszlása az állapotnak. Állapotátmenetek az irányított élek, címkézésük a tüzelések. Egy csomópont esetén legfeljebb annyi rákövetkező csomópont van, ahány engedélyezett tranzíció (kevesebb, ha prioritásos a Petri háló). Holtpont az a csomópont, amiből nem indul ki él. *Nem korlátos Petri háló esetén végtelen sok állapot van*, a korlátos háló véges állapotteret von maga után. Szélességi típusú bejárást végzünk a kiindulási állapotból a tüzelések mentén.

Fedési gráf: végtelen állapottér esetén is jól leírható vele az állapottér. Állapottér bejárása során bejárjuk a M_0,…,M'',…M' trajektóriát. Ha itt M'' kisebb vagy egyenlő M'-vel (ami azt jelenti, hogy M'' állapotban minden helyen kisebb egyenlő annyi token van, mint M' állapotban), M'' fedett állapot M' által! Erősen fedett helyekre speciális szimbólum van, ω a végtelenség kifejezője. Előállítása: fedési fában aznos jelölést reprezentáló csomópontokat összevonjuk.

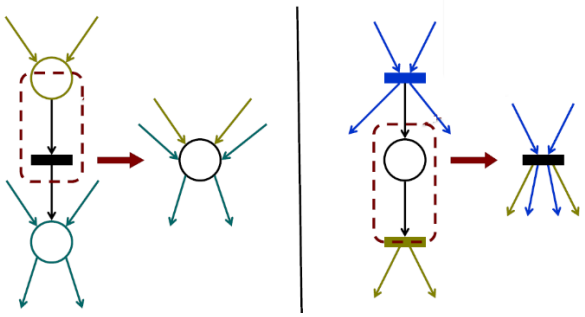
Fedési fa generálása: kezdeti vizsgálandó állapot az M_0. Egy ciklusban, míg a vizsgálandó állapotok halmaza nem üres halmaz, kivesszük a következő gráf csomópontot a vizsgálandó halmazból. Ha ez az M csomópont már előfordult a gyökértől idáig vezető úton, megjelöljük régi csomópontként, és befelyezzük az aktuális iterációt. Ha M még nem fordult elő, de nincs belőle engedélyezett tranzíció, végcsomópontkén jelöljük meg, és befejezzük az aktuális iterációt. Ha van engedélyezett tranzíció, végigiterálunk rajtuk. Meghatározzuk a tranzíció tüzelésével előálló M' csomópontot, és ha van olyan eddigi csomópont, amit M' fed, az M' csomópont tokeneloszlásában az erősen fedett helyeket ω-val jelöljük. Felvesszük a fába M'-t, és egy t címkéjű éllel kötjük össze M-mel. Ezután még hozzávesszük M'-t a vizsgálandó állapotok halmazához, és folytatjuk az M-ben engedélyezett tranzíciókon való végig iterálást.

Leolvasható tulajdonságok:

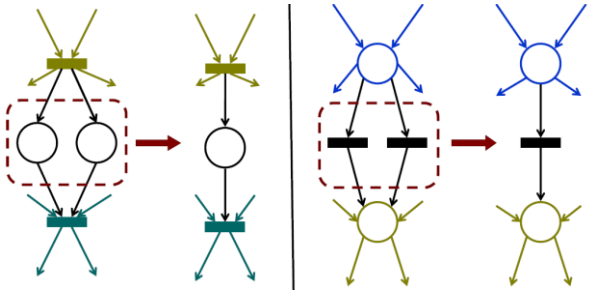
- Petri háló korlátos ⇔ M_0-ból rajzolt elérhetőségi gráf véges ⇔Fedési fában nem jelenik meg a végtelen ω szimbólum
- Petri háló biztonságos ⇔ csak 1 és 0 jelenik meg a csomópont címkéiben a fedési fában
- Petri háló valamely tranzíciója halott (L0-élő) ⇔ tranzícióhoz tartozó tüzelés nem jelenik meg címkeként a fedési fa élein.

Struktúra redukció: célunk az érthetőbb modellből kompakt, nehezebben érthető modellt készíteni, mely a kiválasztott tulajdonságokat megőrzi. A háló **élő, korlátos és biztos** tulajdonságát megőrző transzformációk:

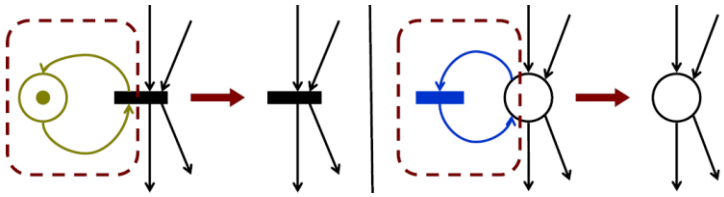
- *Soros összevonások*:



- *Párhuzamos összevonások*:



- *Önhurkok törlése*:



15. Petri hálók strukturális tulajdonságai: Hely és tüzelési invariáns. Invariánsok számítása. Strukturális korlátosság és élőség tulajdonságok definíciója és jellegzetes felhasználásuk.

Tüzelési szám vektor: az egyes tranzíciók tüzeléseinek száma a tüzelési szekvenciában, jele $\boldsymbol{\sigma_T}$.
Engedélyezett σ tüzelési szekvencia esetén a tokeneloszlás módosulása felírható: $\mathbf{M_{i+n}=M_i+W^T\,\sigma_T}$.
Benne lévő információ kevesebb, mint a σ szekvenciában, hisz a sorrend eltűnik. Nem minden σ_T tüzelési szám vektornak megfelelő σ szekvencia végrehajtható adott kezdőállapotból. DE minden σ szekvenciához megadható olyan M_0 állapot, amelyből kezdőállapotként a szekvencia végrehajtható (pl. mindenhova elegendő tokent téve).

Tüzelési invariáns (T-invariáns): σ_T tüzelési szám vektor T-invariáns, ha végrehajtása *nem változtatja meg a tokeneloszlást*. A nekik megfelelő σ szekvenciák végrehajtása esetén ciklus van az állapottérben, M_i [σ > M_i. Ezesetben $\mathbf{W^T\,\sigma_T = 0}$, ennek az egyenletrendszernek a megoldásai adják meg őket. Egy megoldás többszöröse is megoldás (bárhányszor végrehajtva a ciklust), megoldások összege is megoldás (több ciklust kombinálva egymás után), és a megoldások lineáris kombinációi is megoldások. A megoldásokhoz bázis kereshető, ami az összes megoldást előállító minimális halmaz.
Minimális alapú a σ_T T-invariáns, ha nincs olyan T-invariáns, amelynek alapja (**sup(σ)**, azon tranzíciók halmaza, amelyek σ szekvenciában előfordulnak) σ_T alapjának valódi részhalmaza (nem egyenlő), vagy ha részhalmaza azonos, annak tüzelési számai kisebbek.
Alkalmazásuk: folyamatok esetén modell ciklikusságára mutatnak rá, dinamikus tulajdonságok vizsgálatakor a ciklikusan tüzelhető háló egyben megfordítható és visszatérő állapottal is rendelkezik, később is tüzelhető tranzakció pedig az élő tulajdonságot és holtpontmentességet bizonyítja.

Hely invariáns (P-invariáns): A $\boldsymbol{\mu_P}$ nemnegatív súlyvektor által kijelölt *helyek, ahol a tokenek súlyozott összege nem változik* a működés során, $\mu_P{}^T M$ = állandó (tokenek száma a helyek egy részhalmazában állandó, pl. erőforrások). $\mathbf{W\,\mu_P = 0}$ egyenletrendszer adja a megoldásokat.
Alkalmazásuk: folyamatok modellje esetén erőforrások állandóságát mutatja, dinamikus tulajdonságok vizsgálatakor korlátosságra utal, ha token nem termelődik, és láthatjuk, hogy elvész-e token.

Martinez-Silva algoritmus: Végig iterálunk a tranzíciókon. A $\mathbf{Q_i}$ mátrix mindig egy p (p hely van) soros, p+t oszlopos mátrix, melynek első oszlopai egy pxp méretű *egységmátrix*, a többi oszlop pedig a *transzponált szomszédossági mátrix*! Legyen L_p a Q_i mátrix p. sora.
Ezután addig iterálunk a következő műveletekkel, míg a Q_i mátrix jobb oldalán lévő transzponált szomszédossági mátrix nem alakul át nullmátrixszá. Először választunk egy eddig nem vizsgált tranzakciót, és annak oszlopát a szomszédos mátrix transzponáltjában. Felépítünk egy L_{delete} halmazt, Q_{i+1}-ből ezen halmazbeli sorokat fogjuk törölni. Legyen először Q_{i+1} Q_i másolata. A vizsgált tranzakció oszlopában olyan nemnulla értékek párosait keressük, melyekhez létezik két olyan pozitív természetes szám tényező, melyekkel mint súlytényezőkkel összegezve a két számot az eredmény 0! Adjuk hozzá Q_{i+1}-hez ezen sorok súlytényezőkkel való szorzatát, ezzel lenulláztuk a tranzakció oszlopában lévő értékeket, cserébe módosult a bal oldalon lévő egységmátrix is. L_{delete} halmazhoz pedig hozzáadjuk a két sort, ezeket törölni fogjuk, miután minden értéket lenulláztunk a tranzakció oszlopában.
Amikor a jobb oldali részmátrix 0, az egységmátrixból előállt, megmaradt sorok adják a P invariánsokat! Az invariánshoz tartozó tokenösszeg a helyek kezdő tokeneloszlásából adódik.
T-invariánsok keresésekor ugyanígy, de a Q_i mátrix jobb oldalán a szomszédossági mátrixot nem kell transzponálni!

Strukturális élőség: Egy Petri-háló strukturálisan élő, ha létezik olyan M_0 kezdőállapota, amelyben élő. Használat: biztosan végrehajthatóak-e egyes tevékenységek?

Strukturálisan korlátos: ha bármely korlátos M_0 kezdőállapotra korlátos. Használat: Felgyűlnek-e teendők?

16. Színezett Petri hálók: Színezett Petri hálók felépítése. Multihalmazok, kezdőállapot megadása. Élkifejezések, őrfeltételek használata.

Színezett Petri háló: színezetlen hálók kiterjesztései, rugalmas adatszerkezetekkel és adatmanipulációs nyelvvel. A tokeneket színezzük benne. Ötvözik a grafikus reprezentációt (áttekinthetőek), adatmanipulációt (kifejezőereje nagy), és jól definiált szemantikával rendelkezik (formális analízishez megfelelő).

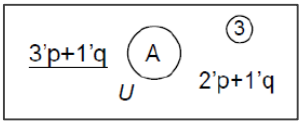
CPN modell = háló struktúra + deklarációk+ háló jelölések, kifejezések + inicializáló kifejezések

Felépítése: *Tokeneket* színezzük, amik adatértékeket reprezentálnak. A színhalmaznak adattípust adhatunk meg, alaptípussal (int, bool, string…) vagy felsorolással (with), vagy komplex is lehet (pl. product U*I).

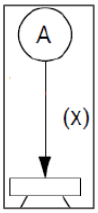
```
color U = with p | q;
color I = int;
color P = product U * I;
color E = with e;
var x : U;
var i : I;
```

Felvehetünk *változókat*, amikhez típust (színhalmazt) rendelhetünk. Tehát hogy milyen token hordozói.
A deklarálást formális nyelven, Standard ML nyelven végezzük, definiálva az adatstruktúrákat és a felhasznált függvényeket.

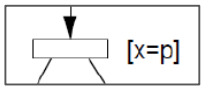
A *helyek* is típussal rendelkeznek, színkészlettel (színhalmaz), ami megadja, milyen típusú tokeneket képes fogadni. A hely mellé írjuk dőlt betűkkel.



Inicializáló kifejezéssel adjuk meg a helyek kezdeti jelölését. A színhalmaz egy multi-halmaz, egy adott színből több token is lehet egy helyen. A hely mellé írjuk, aláhúzva a kifejezést.
Az aktuális jelölését a helynek (tokenek aktuális száma) a hely mellé írjuk, bekarikázva, alatta a részletes megadással, hogy melyik színű tokenből hány van a helyen jelenleg.



A *háló éleire* élkifejezések kerülnek, mely itt is az elveendő és kirakandó tokeneket írják le. Megadják a háló szintaktikai és adatmanipulációs elemeit. Típusa az élhez tartozó hely típusa, és egy tranzícióhoz több típusú él is húzható. Él mellett tüntetjük fel.
Változó használható az élkifejezésben, adatértékeket (színezett tokeneket) lehet hozzá kötni a bementi helyről. Kell legyen típusa, azaz hogy milye színhalmaz elemei köthetőek hozzá, és ha egy tranzíció esetén több élkifejezésben használva van, értelem szerűen ugyanazt az értéket kell felvenniük minden él esetén a tranzíció tüzelésekor. Függvényt is létrehozhatunk és használhatunk élkifejezésekben (f(x)).



A tranzíciók őrfeltételeket kaphatnak, melyek plusz feltételt definiálnak, mikor engedélyezett a tranzíció. Ezek Boole-kifejezések, a tranzíció mellé írjuk, szögletes zárójelben.

17. Színezett Petri hálók működése: Engedélyezettség és tüzelés színezett Petri hálókban. Átalakítás ekvivalens színezetlen hálóvá (széthajtogatás).

Változók lekötése: Színezett tokenek eloszlás a helyek mellett látható.

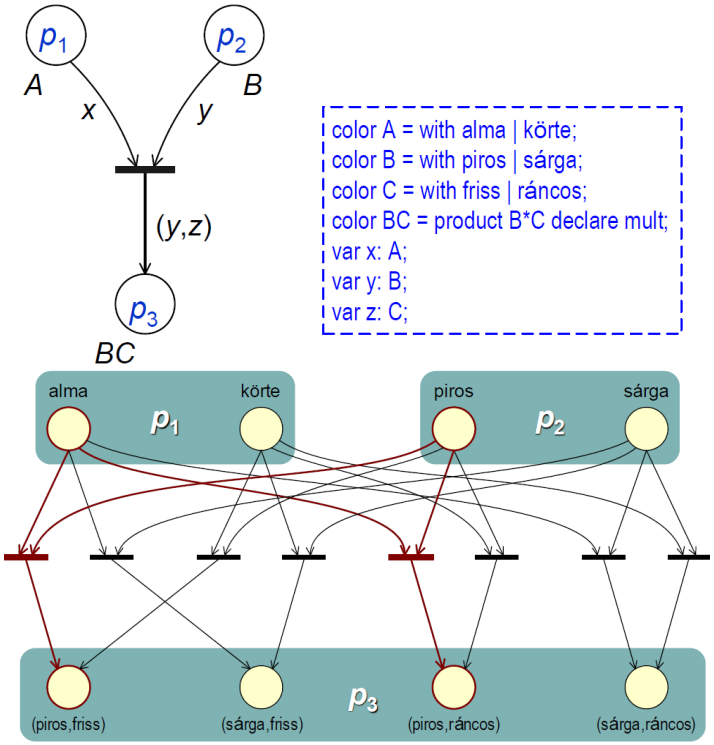
Tranzíció élkifejezéseiben lekötjük a változókat, adatértékez (színezett tokenhez) kötjük a bemeneti helyről. Egy tranzíció esetén egy adott változó minden előfordulása ugyanazt az értéket fogja felvenni. Különböző tranzíciók a lekötés szempontjából függetlenek.
Lekötetlen változó a *kimenő élen* típusának minden értékét felveheti.

Tranzíció engedélyezett egy adott jelölésben egy adott lekötésre, ha a bemenő helyek tartalmazzák azokat a tokeneket, ami az élkifejezés értéke az adott lekötésben, és az őrfeltétel igaz. Ekkor a tranzíció tüzelhet.
Tüzeléshez egy *kötési elem* egy **(tranzíció, lekötés)** pár, pl. (T1, <x=p>). Egy tranzíció esetén több lekötés létezik, ezekből több engedélyezett kötési elem képezhető, amik tüzelhetőek.

Tüzelés egy lekötésben: bemenő helyekről az élkifejezés adott kötésben lévő értéke által meghatározott számú és színű tokenek elvétele, majd kimenő helyekre az élkifejezés adott kötésben lévő értéke által meghatározott számú és színű tokenek odarakása. A lépés hatására a színezett Petri háló egy jelöléséből egy másik lesz.

Széthajtogatás: színezett hálók modellező ereje megfelel a tiltó éllel kiegészített színezetlen hálókénak, minden színezett hálónak megfeleltethető egy ekvivalens működésű színezetlen háló: „széthajtogatott" háló. Széthajtogatásnál a tokenek adattartalmát struktúrában fejezzük ki.
Minden eseménynek (tüzelésnek) a színezett hálóban megfelel egy és csak egy esemény (tüzelés) a széthajtogatott hálóban.

- CPN hely -> több PN hely, a színosztály elemei szerint szétbontva
- CPN tranzíció -> PN tranzíciók a lekötések szerint, minden lehetséges értéknek egy.



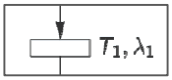
18. Sztochasztikus Petri hálók: Tüzelési szabály sztochasztikus Petri hálókban. Sztochasztikus Petri háló osztályok.

Alapkoncepció, hogy az időt a tranzíciók tüzeléséhez kötjük: a tüzeléssel leírható tevékenység, történés, állapotváltozás idejét modellezzük.

Sztochasztikus Petri háló: minden tranzíciójához tüzelési időt (késleltetést) rendelünk, a tüzelési késleltetés véletlen (valószínűségi változóval írható le, egy adott eloszlás szerint sorsolja a késleltetési időt), és a tüzelési késleltetés statisztikailag független a többi tranzíció késleltetési idejétől.

Osztályok:

- *Sztochasztikus Petri háló (SPN)*: egyszerű Petri hálók kiterjesztése. A tranzíciókhoz véletlen tüzelési késleltetést rendelünk, ami negatív exponenciális valószínűségi eloszlásfüggvénnyel jellemezhető. Az exponenciális eloszlás paramétere, rájáta λ jellel van jelölve a tranzíció mellett. A tranzíciók ÜRES TÉGLALAPPAL vannak jelölve.

  A sorsolt d_i késleltetési időre $\mathbf{P\{d_i<=t\}=1-e^{-\lambda t}}$, $\mathbf{P\{d_i>t\}=e^{-\lambda t}}$.
  Tüzelés szemantikája: engedélyezettség feltétele ugyanaz, mint sima Petri hálóknál. Tüzelési szabály új, akkor tüzelhet a tranzíció a t+d időpillanatban, ha a t időpillanatban engedélyezetté vált, d késleltetési időt sorsolt a hozzá tartozó eloszlásfüggvény szerint, és a [t, t+d) időtartományban folyamatosan engedélyezett volt!
  Konfliktus: ha több tranzíció engedélyezett, az tüzel, amelynek hamarabb letelik a sorsolt késleltetési ideje. Az engedélyezett tranzíciók tehát versenyben vannak, a sorsolt idők alapján hoz döntést.
  Újrasosolás: Tüzelés után az engedélyezett tranzíciók késleltetésének újrasorsolása nem triviális feladat. SPN esetén indifferens, mert a késleltetési idő exponenicális eloszlás miatt fenáll a Markov tulajdonság, az engedélyezett tranzíciók tüzelésig hátralevő ideje ugyanolyan exponenciális eloszlású marad, nem számít, hogy mennyi ideig voltak már engedélyezve. A tüzelésig hátralevő idő statisztikailag független az engedélyezetté válás óta eltelt időtől!
  Az aktuális háló állapot, tokeneloszlás fennmaradásának, jelölés elhagyásának várható ideje a tranzíciók rátájának összegének a reciproka.
- *Általánosított sztochasztikus Petri háló (GSPN)*: SPN kiterjesztett változata, logikai függőségek modellezésére *azonnali tranzíciókkal*, konfliktus feloldására *prioritással* a tranzíciók között, tiltó élekkel és őrfeltételekkel (egyszerűsíthető a háló, élek helyett predikátumok).
- *Determinisztikus és sztochasztikus Petri háló (DSPN)*: További kiterjesztésekkel rendelkező SPN, *determinisztikus késleltetés*sel ellátott tranzíciók is lehetnek, amik konstans késleltetést használnak. Determinisztikus idejű aktivitások modellezésére alkalmas. Fekete vastag téglalappal jelöljük.
  Analízis hatékonyságának feltétele, hogy egyszerre csak egy determinisztikus késleltetésű tranzíció legyen engedélyezett, így az elérhetőségi gráv Markovi analízissel vizsgálható marad.