Temporális logikák és modell ellenőrzés

Előadásvázlat a “Szoftver verifikáció és validáció” tárgyhoz

Majzik István
Budapesti Műszaki Egyetem
Méréstechnika és Információs Rendszerek Tanszék

2004. június 15.

Tartalomjegyzék

1. Bevezető

Az úgynevezett *modális logika* osztályát eredetileg a filozófusok használták kijelentések különböző "módjainak" tanulmányozására. Ilyen módok lehetnek pl. az "esetleg", "mindig", "szükségszerűen", "valamikor biztosan".

A temporális logikák a modális logikák egy formális rendszerét képezik arra, hogy kijelentések igazságának logikai *időbeli* (sorrendiségi) változását vizsgálhassuk. A temporális logika rendszerében *temporális operátorok* állnak rendelkezésre erre a célra. Ilyen operátor lehet például a "*mindig* P", ami akkor igaz, ha a P kijelentés minden jövőbeli időpillanatban igaz; illetve a "*valamikor* Q", ami akkor igaz, ha van olyan jövőbeli időpillanat, amikor a Q kijelentés fennáll.

Hangsúlyoznunk kell, hogy itt az időbeliség tipikusan logikai időre, tehát az időpillanatok sorrendiségére vonatkozik, a valós idő múlását (pl. időtartományok nagysága, időpontok megadása) a temporális logika operátorai általában nem kezelik.

A temporális logikákat elsősorban folyamatosan működő rendszerek (pl. operációs rendszerek, beágyazott rendszerek, egyes protokollok) tulajdonságainak leírására használhatjuk. Ezekben a rendszerekben a bemenetek és a kimenetek kapcsolata nem adható meg transzformációként, és a helyesség sem fogalmazható meg a kezdeti és a végállapotra vonatkozó elő- és utófeltételek formájában (hiszen pl. nem értelmezhető a végállapot). A temporális logika operátorai alkalmasak a folyamatos működés és az ennek kapcsán felmerülő tulajdonságok, követelmények kifejezésére is.

A tulajdonságok egy része *lokális*, tehát egy-egy aktuális időpillanathoz köthető, más részük *elérhetőségi*, azaz a működés során jövőbeli időpillanat(ok)ra vonatkozik. Az elérhetőségi tulajdonságokat szokás a *biztonság* illetve az *élőség* kategóriákba sorolni.

A biztonsági tulajdonságok tipikusan bizonyos veszélyes, nemkívánatos helyzetek elkerülését fogalmazzák meg. Ebből adódóan univerzális kvantort alkalmaznak az időpillanatokra ("minden időpillanatban igaz, hogy a rendszer biztonságos állapotban van"). Általában induktív módszerekkel bizonyíthatók. Ilyen, biztonsági jellegű tulajdonságok például egy többprocesszes rendszer esetén a következők:

- Holtpontmentesség: Minden időpillanatban van futásra kész processz (nincs olyan időpillanat, amikor csak terminált vagy várakozó processz létezik).
- Kölcsönös kizárás: Minden időpillanatban igaz, hogy nincs két vagy több processz egyszerre egy kritikus szakaszban.
- Adatbiztonság: Minden időpillanatban igaz, hogy nincs jogosulatlan hozzáférés (pl. csak olyan adat olvasása történik, amelyhez az olvasó processznek van jogosultsága).

Az élő jellegű tulajdonságok tipikusan bizonyos kívánatos helyzetek elérését írják elő (pl. kérésre válasz érkezik, eredmény előáll). Ezeket az időpillanatokon alkalmazott egzisztenciális kvantorokkal lehet megfogalmazni. Az ellenőrzés nehézsége abból adódik, hogy itt az induktív módszerek nem vethetők be. Általában azt kell megmutatni, hogy a rendszer mindig "közelebb kerül" a kívánatos helyzethez, majd eléri azt. Élő jellegű tulajdonságok például a következők:

- Elküldött üzenet megérkezik: Ha egy üzenetküldés történt, akkor valamikor bekövetkezik az az időpillanat, amikor az üzenet megérkezik.
- Kérés kiszolgálása megtörténik: Egy kérést követően valamikor bekövetkezik az az időpillanat, amikor a kérés kiszolgálása megtörténik.
- Nincs kiéheztetés: Minden processz előbb-utóbb futhat, tehát létezik olyan jövőbeli időpillanat, amikor a processz futó állapotba kerül.

- Terminálás: A program előbb-utóbb eléri a végállapotát.

A tulajdonságok egy része nem sorolható ezekbe a kategóriákba. Ilyen lehet például, ha azt szeretnénk megfogalmazni, hogy egy adott helyzet végtelenül sokszor fennáll (soha nincs olyan időpillanat, amikor azt mondhatnánk, hogy ezentúl az a helyzet soha nem áll fenn).

A fenti tulajdonságok leírására a temporális logikák több osztályát definiálták. Ezekkel foglalkozik a következő fejezet.

2. Temporális logikák osztályozása

A temporális logikák osztályozása többféle kritérium alapján történhet:

Kijelentés- illetve elsőrendű logikák: A temporális kijelentéslogikák – a temporális operátoroktól eltekintve – a klasszikus kijelentéslogika eszköztárát használják. Hasonlóan megfogalmazva, az elsőrendű temporális logikák a temporális operátorok mellett az elsőrendű logikák eszköztárát használják.

Pont- illetve intervallum logikák: A pont logikák jellemzője, hogy a temporális operátorokat egy-egy időpillanatban értékeljük ki. Olykor azonban hasznos lehet intervallum logikát használni, amikor a temporális operátorokat időintervallumokra definiáljuk és értékeljük ki.

Diszkrét- illetve folytonos idejű logikák: Hasonló meggondolások vezethetnek a diszkrét illetve folytonos időkezelés megkülönböztetéséhez. A legtöbb esetben (amikor például programok vagy állapotgépek vizsgálatáról van szó) elégséges az idő diszkrét kezelése: egymás utáni időpillanatokat veszünk figyelembe, amelyek megfeleltethetők a természetes számok sorozatának. Hibrid (pl. analóg elemeket is tartalmazó) és valósidejű rendszerek esetén lehet hasznos a folytonos idő kezelése.

Lineáris illetve elágazó idejű temporális logikák: Az idő (sorrendiség) kezelése szempontjából kétféle esetet különböztethetünk meg. Egyik esetben az egymás utáni időpillanatokat mint egy lineáris rendszert kezeljük: minden időpillanatnak csak egy rákövetkező időpillanata értelmezett (egyféle jövőt veszünk figyelembe). A másik esetben az egymás utáni időpillanatok egy elágazó fa struktúrát alkotnak, minden időpillanatnak többféle lehetséges rákövetkezője értelmezett (többféle lehetséges jövőt figyelembe veszünk).
Az időkezelés alapján a temporális logikák következő osztályait különböztetjük meg:

- Lineáris idejű temporális logika (LTL, *Linear Time Temporal Logic*): A logikai idő szemantikája lineáris, az időpillanatok egy idővonal mentén követik egymást. A temporális operátorok erre az idővonalra vonatkoznak.
- Elágazó idejű temporális logika (BTL, *Branching Time Temporal Logic*): A logikai idő szemantikája elágazó, az időpillanatok fa struktúrában elágazó idővonalak mentén követik egymást. A temporális operátorok az elágazásokra is vonatkoznak (nemcsak egy-egy idővonalra); pl. kifejezhető, hogy minden lehetséges elágazásra igaz valami, illetve legalább egy idővonalra igaz.

Múlt illetve jövő kezelése: A temporális logikák általában a jövő leírására használatosak, hiszen rendszerint a rendszerek kezdőállapotából indulva adjuk meg a specifikációt illetve végezzük el egyes tulajdonságok vizsgálatát. A múlt kezelését az indokolhatja, hogy segítségével néhány tulajdonság egyszerűbben leírható (pl. előző állapotokra, egyes változók előző értékeire lehet hivatkozni).

3. A temporális logikák modelljei

A továbbiakban a jövő időt kezelő, időpillanatokban értelmezett, diszkrét idejű kijelentéslogikákkal foglalkozunk. Ezen temporális logikák segítségével leírt tulajdonságokat (pl. ”az útkereszteződésben a lámpa *valamikor* zöld lesz”) diszkrét állapotokkal illetve akciókkal (műveletekkel) rendelkező rendszereken, például számítógépes programokon vagy állapotgépeken szeretnénk ellenőrizni (pl. a közlekedési lámpa vezérlőjén). A jelen időpillanat az aktuális állapotot (vagy akciót), a jövő időpillanatok pedig a rákövetkező állapotokat (akciókat) jelölik, tehát az egymás utáni időpillanatok az állapotok (akciók) egymásutániságának (szekvenciájának) felelnek meg.

A temporális logikák modelljeiként olyan struktúrákat (formalizmusokat) használunk, amelyek egyszerűek és matematikailag jól kezelhetők. Ilyen struktúrákra fogjuk megadni, milyen módszerekkel lehet egy temporális logikai kijelentés igazságát ellenőrizni. Ezek az alapszintű matematikai struktúrák általában a szemantika alapján származtathatók a mérnöki tervezéshez közelebb álló félformális modellekből (pl. állapottérképből, adatfolyam gráfból) is.

3.1. Kripke-struktúrák

Legyen AP *atomi kijelentések* egy véges halmaza. Ezek az adott alkalmazásban tovább nem bontható kijelentések, pl. ”a lámpa piros”, ”$x > 25$”, ”a processz a kritikus szakaszban van” stb. Az atomi kijelentéseket $P, Q, ...$ nagybetűkkel jelöljük.

Egy adott AP mellett a *Kripke-struktúra* (Kripke-structure, KS) a következő hármas: $M = (S, R, L)$ ahol

- S az állapotok véges halmaza (az állapotokat általában s-sel jelöljük),
- $R \subseteq S \times S$ állapotátmeneti reláció,
- $L : S \to 2^{AP}$ az állapotok címkézése az atomi kijelentésekkel. Egy állapotot több kijelentés is címkézhet. Minden s állapotra $\mathrm{true} \in L(s)$ és $\mathrm{false} \notin L(s)$, ahol true az ”igaz” kijelentés (mindenütt igaz), false pedig a ”hamis” (sehol sem igaz).

A Kripke-struktúra teljes, ha $\forall s \in S : \exists t \in S, (s, t) \in R$ (egyébként részlegesnek mondjuk).

Egy példa látható a 1. ábrán. Itt $AP = \{$Zöld, Sárga, Piros, Villogó_sárga$\}$, és pl. $L(s_4) = \{$Piros, Sárga$\}$.

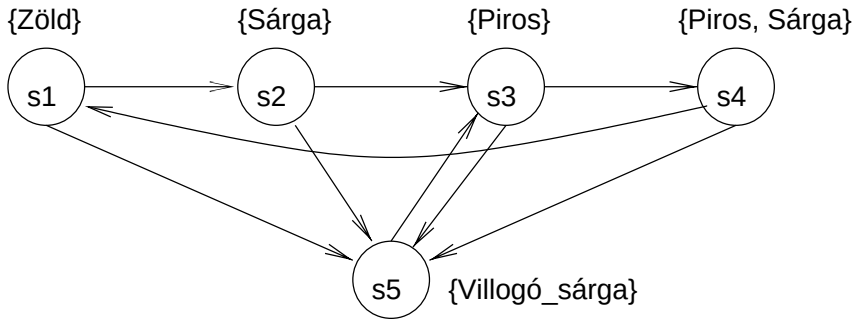
Kripke-struktúrákat használunk akkor, ha a rendszerünk működését az állapotainak megadásával tudjuk a legjobban leírni, az egyes állapotokat pedig lokálisan, az adott állapotban igaz kijelentésekkel tudjuk jellemezni. Tipikusan ilyenek a jól megfogható állapotváltozókkal rendelkező rendszerek, hiszen az egyes állapotok az állapotváltozók értékeivel (vagy értéktartományaival) mint kijelentésekkel jellemezhetők. A temporális logika segítségével leírt rendszerszintű tulajdonság igazságát az egyes állapotokon érvényes lokális kijelentések alapján értékeljük ki.

3.2. Címkézett állapotátmeneti rendszerek

A címkézett állapotátmeneti rendszerek vagy tranzíciós rendszerek (Labelled Transition System, LTS) esetén az állapotátmenetekhez ún. *akciókat* rendelünk. Az akciók tovább nem bonthatók, általában egy-egy (alkalmazás-specifikus) üzenetet, műveletet, a környezettel való valamilyen kölcsönhatást jelentenek. Az akciókat az $a, b, c, ...$ kisbetűkkel jelöljük.

Egy LTS a következő hármas: $T = (S, Act, \to)$, ahol

- S az állapotok véges halmaza (az állapotokat általában s-sel jelöljük),

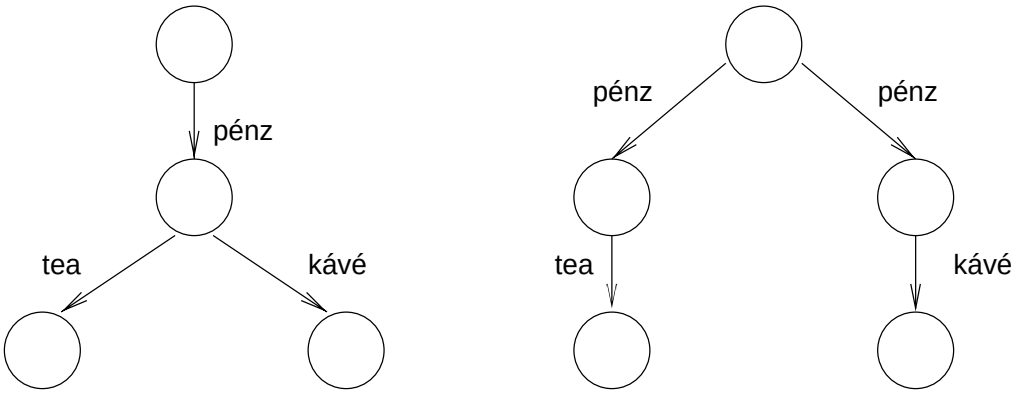


1. ábra. Egy Kripke-struktúra

- $Act = \{a, b, c, ...\}$ az akciók véges halmaza,
- $\rightarrow \subseteq S \times Act \times S$ címkézett állapotátmeneti reláció. Egy állapotátmenetet egy akció címkézhet. Egy s és s' állapotok közötti, a akcióval címkézett állapotátmenet szokásos jelölése $s \xrightarrow{a} s'$.

LTS-ekre mutat példát a 2. ábra, ahol egy italautomata kétféle modellje látható.

LTS modelleket használunk akkor, ha a rendszerünk működését leginkább az állapotátmenetek során bekövetkező akciók sorozatával tudjuk leírni (az egyes állapotokat pedig kevésbé tudjuk lokálisan jellemezni). Tipikusan ilyenek a kommunikáló (üzenetet küldő és fogadó) rendszerek. A temporális logika segítségével leírt rendszerszintű tulajdonság igazságát a lehetséges akciósorozatok alapján értékeljük ki.



2. ábra. LTS-ek

3.3. Kripke állapotátmeneti rendszerek

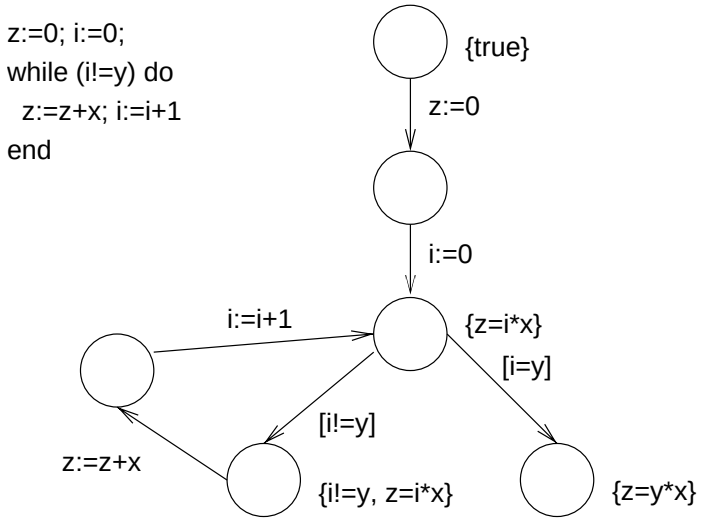
Kripke állapotátmeneti rendszerek (Kripke Transition System, KTS) esetén az állapotokat kijelentésekkel, az átmeneteket pedig akciókkal címkézhetjük. Adott AP és Act mellett

$K = (S, \longrightarrow, L)$ ahol az egyes betűk jelentése az előző alfejezetek alapján már világos.

KTS modelleket használhatunk például programok esetén a programutasítások (az átmenetekhez rendelt akciók) és a programváltozók (az állapotokhoz rendelt kijelentések) egyidejű megadására. Egy ilyen KTS modellre mutat példát a 3. ábra.

3.4. Automaták véges szavakon

Véges hosszúságú szavakon értelmezhetjük az $A = (\Sigma, S, S_0, \rho, F)$ automatát, ahol



3. ábra. Egy program és az azt leíró KTS

- Σ az ábécé (betűk nemüres halmaza),
- S az állapotok véges, nemüres halmaza,
- $S_0 \subseteq S$ a kezdőállapotok halmaza,
- $\rho : S \times \Sigma \to 2^S$ az állapotátmeneti reláció (egy beérkező betű hatására új állapotba lép az automata),
- F az elfogadó állapotok halmaza.

Egy A automata determinisztikus, ha $|S_0| = 1$, valamint $\forall s \in S,\ \forall a \in \Sigma : |\rho(s, a)| \leq 1$.

Az A automata *futása* egy beérkező $a_0, a_1, a_2, ..., a_{n-1}$ betűsorozat (szó) hatására az $r = (s_0, s_1, s_2, ..., s_n)$ állapotsorozat, ahol $s_0 \in S_0$, $s_{i+1} = \rho(s_i, a_i)$, $0 \leq i < n$. A futást elfogadónak nevezzük ha $s_n \in F$. Egy $w = (a_0, a_1, a_2, ..., a_{n-1})$ szót elfogad az automata, ha létezik rá elfogadó futás.

Az automata által elfogadott nyelv $L(A) = \{w \in \Sigma^* \mid w \text{ elfogadott}\}$.

A véges szavakon értelmezett automatákat használhatjuk pl. véges hosszú bemenetek feldolgozásának leírására. A temporális logika segítségével leírt rendszerszintű tulajdonság igazságát az elfogadott nyelv alapján (hasonlóan, mint LTS esetén a lehetséges akciósorozatok alapján) értékeljük ki.

3.5. Büchi-automaták

A Büchi-automatákat végtelen hosszúságú szavakon értelmezzük. Ez esetben módosítanunk kell az elfogadás kritériumát, hiszen nincs végállapot.

Az A automata *futása* egy beérkező $a_0, a_1, a_2, ...$ végtelen betűsorozat (szó) hatására az $r = (s_0, s_1, s_2, ...)$ állapotsorozat, ahol $s_0 \in S_0$, $s_{i+1} = \rho(s_i, a_i)$, $0 \leq i$.

A végtelen hosszúságú futás jellemzője azon $s \in S$ állapotok halmaza, amelyeket a futás végtelenül sokszor érint:

$\lim(r) = \{s \mid s = s_i$ végtelenül sokszor, azaz nincs olyan j, hogy $\forall k > j : s \neq s_i\}$

Egy futást elfogadónak nevezünk, ha $\lim(r) \cap F \neq \emptyset$. Egy w végtelen hosszúságú szót elfogad az automata, ha létezik rá elfogadó futás.

A Büchi-automata által elfogadott nyelv $L(A) = \{w \in \Sigma^\omega \mid w \text{ elfogadott}\}$.

A temporális logika segítségével leírt rendszerszintű tulajdonság igazságát itt is az elfogadott nyelv alapján értékeljük ki. A Büchi-automaták lehetőséget adnak folyamatosan működő rendszerek leírására.

4. Lineáris idejű temporális logikák

Az alábbiakban ismertetendő logika egy lineáris idejű temporális kijelentéslogika (PLTL, Propositional LTL). A PLTL kifejezéseket Kripke-struktúrák egy-egy útvonalán értelmezzük. Az útvonalakat szokás a PLTL időstruktúrájának is nevezni: az egymás utáni állapotok felelnek meg az idővonalon található időpillanatoknak. Létezik a rendszerindításnak megfelelő kezdő időpillanat, a jövő pedig végtelen (ha folyamatosan működik a rendszer). Az egymást követő időpillanatok helyett ezentúl az egymást követő állapotokra fogunk hivatkozni.

4.1. Temporális operátorok

A PLTL $p, q, r, \ldots$ kifejezései a következő elemekből épülnek fel:

- Atomi kijelentések, amelyek tovább nem bonthatók: $P, Q, R \ldots$ (pl. “a lámpa zöld”, “$x = 25$”, “a rendszer az $s2$ állapotban van”). Ezek az atomi kijelentések ”címkézik” a rendszer állapotait, ezek határozzák meg azt az absztrakciós szintet, ahol a tulajdonságokat értelmezzük.
- Boole logikai operátorok: $\wedge$ (ÉS kapcsolat), $\vee$ (VAGY kapcsolat), $\neg$ (negálás).
- Temporális operátorok.

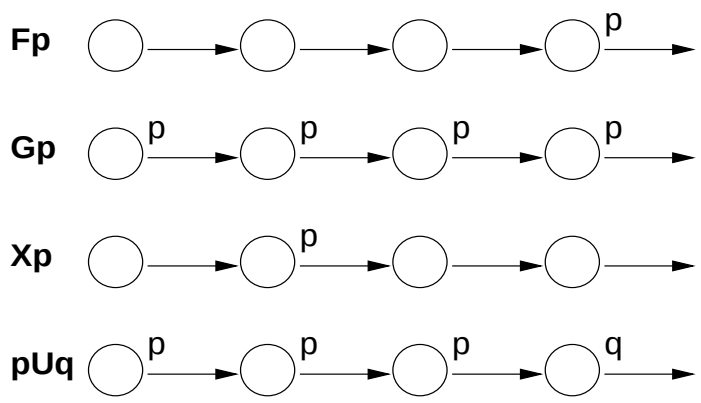
A PLTL-ben az alábbi temporális operátorokat használjuk a p illetve q kifejezésekre vonatkozóan:

- $F\,p$: valamikor p; egy jövőbeli állapotban igaz lesz p. A jelölés megjegyezhető az angol ”Future” szó alapján.
- $G\,p$: mindig p; minden jövőbeli állapotban igaz lesz p. A jelölés itt az angol ”Globally” szó alapján jegyezhető meg.
- $X\,p$: következő p; a következő állapotban igaz lesz p. Itt az angol ”neXttime” használható emlékeztetőként.
- $p\,U\,q$: p amíg q; egy jövőbeli állapotban igaz lesz q, addig pedig minden állapotban igaz lesz p. A jelölés az angol ”Until” szóra utal.

A temporális operátorok intuitív jelentését egy-egy útvonal (idővonal) ábrázolásával a 4. ábra mutatja be.

Az alábbiakban felírunk néhány PLTL kifejezést és megpróbáljuk megfogalmazni ezek jelentését egy adott útvonalon:

- $p \Rightarrow F\,q$: ha p jelenleg igaz, akkor a jövőben valamikor q is igaz lesz (például p kérésre q válasz érkezik).
- $G(p \Rightarrow F\,q)$: minden állapotban fennáll, hogy ha p igaz, akkor a jövőben valamikor q is igaz lesz (például bármikor kiadva egy p kérést, arra előbb-utóbb q válasz érkezik).
- $p\,U\,(q \vee r)$: p igaz, amíg q vagy r igaz nem lesz (például egy p kérést kiadva arra előbb-utóbb vagy egy q helyes, vagy egy r helytelen válasz érkezik).



4. ábra. Az LTL operátorainak intuitív jelentése

- $\neg((\neg p)\,U\,q)$: nem igaz az, hogy $\neg p$ teljesül amíg q igaz nem lesz; vagyis p legalább egyszer igaz q előtt (például a q választ megelőzi egy p kérés), vagy q igaz sem lesz.
- $GF\,p$: minden állapotban igaz, hogy a jövőben előbb-utóbb p igaz lesz; tehát nem találunk olyan állapotot, ami után p soha nem lesz igaz (például minden állapotból a p tulajdonságú kezdőállapotba vihető a rendszer).
- $FG\,p$: a jövőben olyan állapotba kerül a rendszer, ami után p mindig igaz lesz (például a kezdeti tranziensek után a p tulajdonságú üzemi állapotokba kerül a rendszer).
- $(p \wedge G(p \Rightarrow Xp)) \Rightarrow Gp$ a matematikai indukció formalizálása, mindig igaz kifejezés.

Láthatjuk, a természetes nyelvi megfogalmazás sokszor nehézkes és félreérthető.

A temporális operátorok fenti, informális definíciója is kérdéseket vet fel: például $p\,U\,q$ igaz-e, ha q rögtön az első állapotban igaz? Az operátorok jelentését azok formális szintaxisának és szemantikájának megadásával tehetjük világossá.

4.2. A PLTL formális szintaxisa

A PLTL formális szintaxisát a kifejezések képzési szabályaival adjuk meg. A kifejezések halmaza az alábbi módon képezhető:

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

Mint látjuk, több, már bevezetett operátor illetve logikai művelet nem szerepel a fenti szabályok között. Ezeket mint rövidítéseket definiálhatjuk:

- $p \vee q$ 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 $\mathrm{true}\,U\,p$,
  $G\,p$ jelentése $\neg F \neg p$.

Bevezethetünk néhány további, gyakran használt rövidítést is:

- $p\,B\,q$ jelentése $\neg((\neg p)Uq)$,
  $F^{\infty}p$ jelentése $GF\,p$ (végtelen sokszor),
  $G^{\infty}p$ jelentése $FG\,p$ (csaknem mindenütt).

Az egyes operátorok precedenciája növekvő sorrendben a következő:
$\equiv, \Rightarrow, \vee, \wedge, \neg, (F, G, X, U)$.

4.3. A PLTL formális szemantikája

Egy p PLTL kifejezést az $M = (S, R, L)$ Kripke-struktúra egy $\pi = (s_0, s_1, s_2, s_3, ...)$ útvonalán értelmezünk, ahol s_0 a kezdőállapot és $i \geq 0 : (s_i, s_{i+1}) \in R$. A következő jelölést használjuk:

- $M, \pi \models p$ jelentése: az M struktúrában a π útvonalon p igaz (itt M jelölése elhagyható, ha az egyértelmű).
- π^i jelentése a π útvonal (állapotsorozat) i-edik elemétől kezdődő része: $\pi^i = (s_i, s_{i+1}, s_{i+2}, ...)$

Ezek alapján megadható a formális szintaxis (L1)-(L3) szabályaiban szereplő operátorok szemantikája (a következőkben az a.cs.a. rövidítés jelentése: ”akkor és csakis akkor, ha”):

- (L1) $\pi \models P$ a.cs.a. $P \in L(s_0)$, ahol P egy atomi kijelentés.
- (L2) $\pi \models p \wedge q$ a.cs.a. $\pi \models p \wedge \pi \models q$,
  $\pi \models \neg p$ a.cs.a. $\pi \models p$ nem igaz.
- (L3) $\pi \models (p\,U\,q)$ a.cs.a. $\exists j : (\pi^j \models q$ valamint $\forall k < j : \pi^k \models p)$,
  $\pi \models Xp$ a.cs.a. $\pi^1 \models p$.

Vegyük észre, hogy $X\,p$ esetén a p kifejezést a kezdőállapotot követő állapotból induló útvonalon értékeljük ki.

A formális szemantika alapján már választ tudunk adni az egyik előző alfejezetben feltett kérdésre: $p\,U\,q$ igaz, ha q rögtön az első állapotban igaz. Azt is látszik, hogy $p\,U\,q$ nem lehet igaz, ha q sohasem következik be.

A teljesség kedvéért a többi operátor induktív szemantikáját is megadjuk (bár ezek felírhatók az előző operátorok segítségével):

- $\pi \models F\,p$ a.cs.a. $\exists j : \pi^j \models p$.
- $\pi \models G\,p$ a.cs.a. $\forall j : \pi^j \models p$.
- $\pi \models (p\,B\,q)$ a.cs.a. $\forall j$: $(\pi^j \models q$ következménye $\exists k < j : \pi^k \models p)$.
- $\pi \models F^{\infty}p$ a.cs.a. $\forall k : \exists j \geq k : \pi^j \models p$.
- $\pi \models G^{\infty}p$ a.cs.a. $\exists k : \forall j > k : \pi^j \models p$.

4.4. Tulajdonságok megadása PLTL kifejezésekkel

A PLTL kifejezések használatára nézzünk egy egyszerű példát: vezetőválasztást egy elosztott rendszerben. A rendszerben szereplő egységeket az $1, 2, ..., N$ indexekkel jelöljük. Az elosztott rendszerben szükség van egy "vezetőre", amely egység speciális feladatot lát el: pl. a többieket koordinálja, órákat szinkronizál, üzenetet sorrendez stb. Ha egy állapotban az i egység a vezető, akkor ezt a leader_i kijelentéssel (címkézéssel) adjuk meg. Egyszerre csak egy vezető lehet a rendszerben, a vezetők választása az index szerinti prioritási sorrendben történik: mindig a legnagyobb indexű egység kell legyen a vezető. Az egységek fokozatosan csatlakoznak a rendszerhez. Ha egy i egység már csatlakozott, akkor a további állapotokat az active_i kijelentéssel címkézzük. Ha egy új egység csatlakozik a rendszerhez, akkor az indexének megfelelően sor kerülhet új vezető választására. Egy új egység addig nem veheti át a vezető szerepet, míg a régi erről le nem mondott. Tételezzük fel, hogy az egységek nem esnek ki a rendszerből.

A helyes vezetőválasztás követelményeit PLTL kifejezésekkel fogjuk megadni. A kifejezésekben használni fogjuk az egzisztenciális és az univerzális kvantorokat, amelyek nem részei a PLTL-nek. Ezért tekintsük ezeket mint egyszerűsítő jelöléseket, amelyek egy rögzített N egységből álló rendszerben feloldhatók a Boole-logika operátoraival: $\forall i : P_i$, ahol P_i az i egységre vonatkozó atomi kijelentés, feloldható $P_1 \wedge P_2 \wedge ... \wedge P_N$ alakban; hasonlóan $\exists i : P_i$ is. Egy másik trükk, hogy a prioritások (mint természetes számok) közötti kisebb-nagyobb relációkat is atomi kijelentésekként fogalmazzuk meg. Ezek valamennyi állapotot címkézik.

A rendszerben a fentiek alapján a következő atomi kijelentések használhatók:

$AP = \{\mathrm{leader}_i,\ \mathrm{active}_i,\ i < j\}$ ahol $0 < i,\ j \leq N$ illetve $i < j$ (tehát csak olyan $i < j$ atomi kijelentések léteznek, ahol az indexekre mint természetes számokra $i < j$ fennáll).

A követelményeket az alábbiakban fogalmazzuk meg:

- *Mindig van a rendszerben egy vezető*:
  $G(\exists i : \mathrm{leader}_i \wedge (\forall j \neq i : \neg \mathrm{leader}_j))$
  Ez a kifejezés jónak tűnik, azonban valójában nem azt fedi le, amit mi szeretnénk. Mivel a vezetőválasztás általában nem "nulla idő alatt", egy oszthatatlan műveletben zajlik le, ezért vannak olyan időpillanatok (rendszerállapotok) választás közben, amikor pl. nincs vezető a rendszerben. De ez csak időlegesen történhet meg.
  Ezért először arra gondolhatunk, hogy a $GF(\exists i : \mathrm{leader}_i \wedge (\forall j \neq i : \neg \mathrm{leader}_j))$ lesz a jó megfogalmazása a követelménynek. Igen ám, de ez a kifejezés azt is megengedi, hogy több vezető legyen egyidejűleg, ami még időlegesen sem engedhető meg (a csatlakozó egység addig nem veheti át a vezető szerepet, míg a régi erről le nem mondott).
  Ezek alapján érdemes két részre bontani a követelmény:

  - *Egyszerre csak egy vezető lehet a rendszerben:*
    $G(\mathrm{leader}_i \Rightarrow \forall j \neq i : \neg \mathrm{leader}_j)$
  - *Előbb-utóbb sikerül vezetőt választani:*
    $GF(\exists i : \mathrm{leader}_i)$

- *Ha belép egy nagyobb indexű egység, akkor a jelenlegi vezetőnek le kell mondania:*
  $G(\forall i, j : ((\mathrm{leader}_i \wedge i < j \wedge \neg \mathrm{leader}_j \wedge \mathrm{active}_j) \Rightarrow F\, \neg \mathrm{leader}_i))$

- *Az új vezető indexe mindig nagyobb, mint az előző vezető indexe:*
  $G(\forall i, j : (\mathrm{leader}_i \wedge \neg X\, \mathrm{leader}_i \wedge XF\, \mathrm{leader}_j) \Rightarrow i < j)$

A tulajdonságok (követelmények) PLTL kifejezések formájában való megadásának – a matematikai precizitás mellett – előnye, hogy nem befolyásolja magának a vezetőválasztási protokollnak a megva-

lósítását. A megvalósítással kapcsolatban nem kell előfeltételezésekkel élnünk, csak az elvárt tulajdonságokra kell koncentrálnunk.

4.5. A PLTL kiterjesztése KTS-re

A PLTL egyszerűen kiterjeszthető KTS-re, ha a PLTL kifejezésekben megengedjük az $a \in Act$ akciók használatát. Az útvonalakat ekkor $\pi = (s_0, a_1, s_1, a_2, s_2, ...)$ alakban írhatjuk, ahol $\forall i \geq 0 : s_i \overset{a_{i+1}}{\rightarrow} s_{i+1}$.

A PLTL szintaxis a következőképpen módosul:

- (L4) Ha a egy akció, akkor (a) egy PLTL kifejezés.

A hozzá kapcsolódó szemantikai szabály:

- (L4) $\pi \models (a)$ a.cs.a. $a_1 = a$ (itt a_1 az első akció π-ben).

5. Elágazó idejű temporális logikák: CTL és CTL*

Az elágazó idejű temporális kijelentéslogikák közül a két legelterjedtebbel, a CTL és a CTL* logikákkal fogunk foglalkozni.

Ezek a logikák egy $M = (S, R, L)$ Kripke-struktúra állapotain értelmezettek. A kezdőállapotból indulva az elágazó idővonalak mentén az egymás utáni állapotokat egy fa-szerű struktúrában ábrázolhatjuk: minden állapotnak legalább egy utódja, rákövetkező állapota lehet. Ez a számítási fa, amiről a logikák a nevüket kapták: CTL a Computational Tree Logic rövidítése.

5.1. A CTL és CTL* temporális operátorai

A CTL és CTL* temporális operátorait két csoportra oszthatjuk. Egyik csoportot az állapotokból induló útvonalak kvantorai, a másik csoportot pedig az útvonalakon az állapotok kvantorai alkotják.

A kvantorok jelölése a következő:

Útvonalak kvantorai: Két új kvantorral találkozunk:

- A: minden lehetséges útra az adott állapotból indulva (”for All futures”).
- E: legalább egy útra az adott állapotból indulva (”Exists a future”).

Állapotok kvantorai: Itt a PLTL-ből már ismerős operátorokat használhatjuk

- G: minden állapotra az adott útvonalon.
- F: legalább egy állapotra valamikor az adott útvonalon.
- X: a következő állapotra az adott útvonalon.
- U: amíg egy következő feltétel igaz nem lesz az adott útvonalon.

A CTL logika esetén kikötés, hogy az útvonal kvantorát mindig közvetlenül követi egy, az útvonal állapotaira vonatkozó PLTL kvantor (ld. később a formális szintaxist). A CTL* a CTL-nél kifejezőbb logika (gyakran úgy hivatkoznak rá, mint ”teljes elágazó idejű temporális logika”), mert az útvonalra vonatkozó kvantort tetszőleges PLTL formula követheti, megengedett Boole-logika és egymásba ágyazás G, F, X és U fölött.

A továbbiakban a CTL* majd a korlátozottabb CTL szintaxisával foglalkozunk, majd a szemantikákat adjuk meg.

5.2. A CTL* formális szintaxisa

A CTL* kifejezések szintaxisát induktívan definiáljuk, állapotokra vonatkozó és útvonalakra vonatkozó rész-kifejezéseket megadva:

Állapot-kifejezések: Az állapot-kifejezések állapotokra vonatkoznak.

- (S1) Minden P atomi kijelentés egy állapot-kifejezés.
- (S2) Ha p és q állapot-kifejezések, akkor $p \wedge q$ és $\neg p$ is.
- (S3) Ha p egy útvonal-kifejezés akkor $E\,p$ és $A\,p$ állapot-kifejezések.

Útvonal-kifejezések: Az útvonal-kifejezések útvonalakra vonatkoznak.

- (P1) Minden állapot-kifejezés egyben egy útvonal-kifejezés is.
- (P2) Ha p és q útvonal-kifejezések, akkor $p \wedge q$ és $\neg p$ is.
- (P3) Ha p és q útvonal-kifejezések, akkor Xp és pUq is.

A fenti szabályok alapján generált *állapot-kifejezések* adják a CTL* nyelvet. (Az itt nem felsorolt operátorok a szokásos módon származtathatók.)

5.3. A CTL formális szintaxisa

A CTL annyiban korlátozott CTL*-hoz képest, hogy a fenti (P1)-(P3) szabályok helyett csak egyetlen P0 szabály szerepel:

- (P0) Ha p és q állapot-kifejezések, akkor Xp és pUq útvonal-kifejezések.

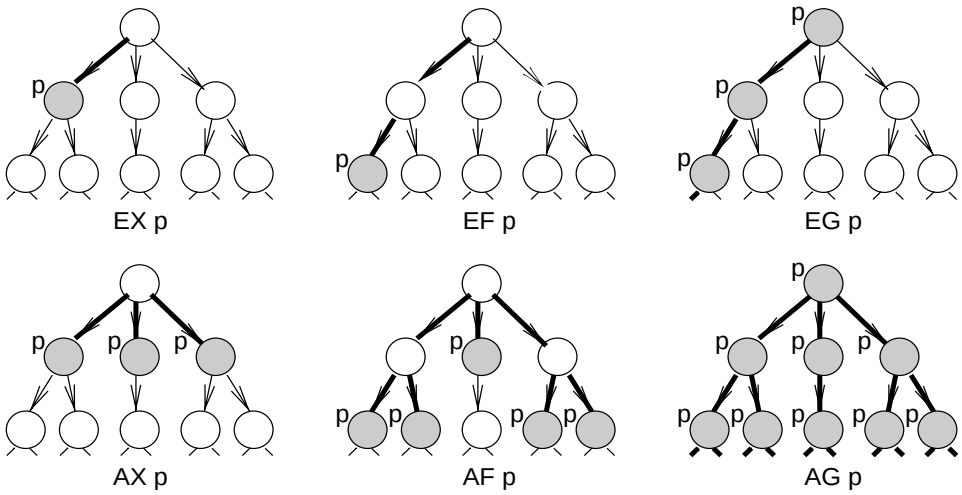
A legfontosabb különbség tehát az, hogy útvonal-kifejezés csak közvetlenül állapot-kifejezésből állítható elő az X és U operátorok segítségével. Útvonal-kifejezések nem ágyazhatók egymásba és nem használhatók Boole-operátorok sem köztük. Az így előállított útvonal-kifejezésekből az E és A operátorokkal képezhetünk állapot-kifejezéseket. Így az E, A valamint az X, U operátorok "össze kell nőjenek", így gyakorlatilag EX, AX, $E(\ U\ )$, $A(\ U\ )$ összetett operátorok állnak elő.

Egy $E(XXX(\mathrm{piros}))$ kifejezés egy helyes CTL* kifejezés (azt írja le, hogy van egy olyan, sorrendben harmadik elérhető állapot, ahol a "piros" atomi kijelentés igaz), de nem helyes CTL-ben, mert itt útvonal-kifejezések vannak egymásba ágyazva. Hasonlóan $E(X(\mathrm{piros}) \vee F(\mathrm{fekete}))$ helyes CTL*-ban, de nem helyes CTL-ben, mert itt útvonal-kifejezések között szerepelnek Boole-operátorok.

Tehát CTL kifejezések az előbbi (S1)-(S3) szabályok és P0 alapján képezhetők. A formális szintaxisban nem szereplő többi Boole-operátor kifejezése a szokásos, míg a szabályokból hiányzó temporális operátorok mint rövidítések foghatók fel:

- EFp jelentése $E(\mathrm{true}Up)$.
- AGp jelentése $\neg EF\neg p$.
- AFp jelentése $A(\mathrm{true}Up)$.
- EGp jelentése $\neg AF\neg p$.

A CTL operátorok intuitív jelentésére az 5 ábra utal.
Az alábbiakban néhány példát mutatunk CTL kifejezésekre, azok (intuitív) jelentésével.



5. ábra. A CTL temporális operátorainak intuitív jelentése

- $AG\,p$: p mindig igaz, azaz $\neg p$ soha nem történhet meg.
- EFp: a kezdeti állapotból indulva lehetséges egy olyan állapotot elérni, ahol p igaz.
- AFp: a kezdeti állapotból indulva mindenképpen elérünk egy olyan állapotot, ahol p igaz.
- $AG\,EF\,p$: bármely állapotból indulva lehetséges egy olyan állapotot elérni, ahol p igaz (pl. bárhonnan alapállapotba vihető a rendszer).
- $AG\,AF\,p$: bármely állapotból indulva mindenképpen elérünk egy olyan állapotba, ahol p igaz lesz.
- $AF\,AG\,p$: $\neg p$ csak véges számban fordulhat elő (a rendszer előbb-utóbb olyan állapotba kerül, ami után p mindig igaz lesz).
- $EF\,AG\,p$: lehetséges, hogy a rendszer olyan állapotba kerül, ami után p mindig igaz lesz.
- $AG(p \Rightarrow AFq)$: minden állapotra fennáll, hogy ha p igaz, akkor valamikor q be fog következni (pl. kérésre válasz érkezik, elkezdett számítás befejeződik).

5.4. A CTL* és CTL szemantikája

Egy CTL* kifejezés szemantikája az $M = (S, R, L)$ Kripke-struktúrán adható meg. Az alábbi jelöléseket fogjuk használni:

- $\pi = (s_0, s_1, ...)$ egy teljes útvonal jelölése, ahol $\forall i \geq 0 : (s_i, s_{i+1}) \in R$.
- π^i jelentése a π útvonal i-edik elemétől kezdődő része: $\pi^i = (s_i, s_{i+1}, s_{i+2}, ...)$.
- $M, s \models p$ (illetve $M, \pi \models p$) jelentése az, hogy a p állapot-kifejezés (illetve p útvonal-kifejezés) igaz az M struktúrában, az s állapotban (illetve a π teljes útvonalon). Ha ez egyértelmű, M jelölése elmaradhat. Külön jelölés nélkül általában M kezdőállapotára vonatkoztatjuk a CTL* és CTL kifejezéseket.

Ez alapján az operátorok szemantikája a szintaxis szabályaihoz kapcsolódva a következőképpen adható meg az állapot-kifejezések esetén:

- (S1) $M, s \models P$ a.cs.a. $P \in L(s)$.
- (S2) $M, s \models p \wedge q$ a.cs.a. $M, s \models p \ \wedge \ M, s \models q$.
  $M, s \models \neg p$ a.cs.a. nem igaz $M, s \models p$.
- (S3) $M, s \models E\,p$ a.cs.a. létezik egy $\pi = (s_0, s_1, ...)$ teljes útvonal M-ben $s = s_0$ mellett, hogy $M, \pi \models p$.
  $M, s \models A\,p$ a.cs.a. minden $\pi = (s_0, s_1, ...)$ teljes útvonalra M-ben, ahol $s = s_0$, fennáll $M, \pi \models p$.

Az útvonal-kifejezések esetén:

- (P1) $M, \pi \models p$ a.cs.a. $M, s_0 \models p$ (itt p állapot-kifejezés, lásd a szintaxis (P1) szabályát).
- (P2) $M, \pi \models p \wedge q$ a.cs.a. $M, \pi \models p$ and $M, \pi \models q$.
  $M, \pi \models \neg p$ a.cs.a. nem igaz $M, \pi \models p$.
- (P3) $M, \pi \models p\,U\,q$ a.cs.a. $\exists i : (M, \pi^i \models q$ és $\forall j < i : M, \pi^j \models p))$.
  $M, \pi \models Xp$ a.cs.a. $M, \pi^1 \models p$.

Egy CTL kifejezés szemantikája ugyanezekkel a szabályokkal kapható, a (P3) szabályt értelemszerűen módosítva a szintaxis (P0) szabályának megfelelően (ott p és q állapot-kifejezések):

- (P0) $M, \pi \models p\,U\,q$ a.cs.a. $\exists i : (M, s_i \models q$ és $\forall j < i : M, s_j \models p))$.
  $M, \pi \models Xp$ a.cs.a. $M, s_1 \models p$.

5.5. Tulajdonságok megadása CTL kifejezésekkel

Példaként tekintsünk egy két processzből (P_1 és P_2) álló rendszert. Egy-egy processz lehet kritikus szakaszban (jelöljük C-vel), nemkritikus szakaszban (N) illetve a kritikus szakaszba való belépésre készen (W). A rendszer állapotait a processzek tulajdonságai alapján a C_1, C_2, N_1,N_2, W_1, W_2 kifejezésekkel címkézzük. A processzek nemkritikus szakaszban indulnak, majd próbálnak belépni a kritikus szakaszba, a belépés után egy idővel elhagyják a kritikus szakaszt és kezdődik minden elölről. Egyszerre csak egy processz lehet a kritikus szakaszban, és a két processz váltja egymást.

Formálisan tehát az atomi kijelentések a következők lehetnek:
$AP = \{C_1, C_2, N_1, N_2, W_1, W_2\}$.
A rendszer tulajdonságai a következőképpen fogalmazhatók meg CTL kifejezések segítségével:

- *Egyszerre csak egy processz lehet a kritikus szakaszban*:
  $AG(\neg(C_1 \wedge C_2))$
- *Ha egy processz be akar lépni a kritikus szakaszba, akkor előbb-utóbb mindenképpen beléphet*:
  $AG(W_1 \Rightarrow AF(C_1))$ és hasonlóan P_2-re is.
- *A processzek váltva lépnek be a kritikus szakaszba*:
  $AG(C_1 \Rightarrow A(C_1\,U\,(\neg C_1 \wedge A(\neg C_1\,U\,C_2))))$ és hasonlóan P_2-re is.

5.6. Kifejezőképesség és méltányosság

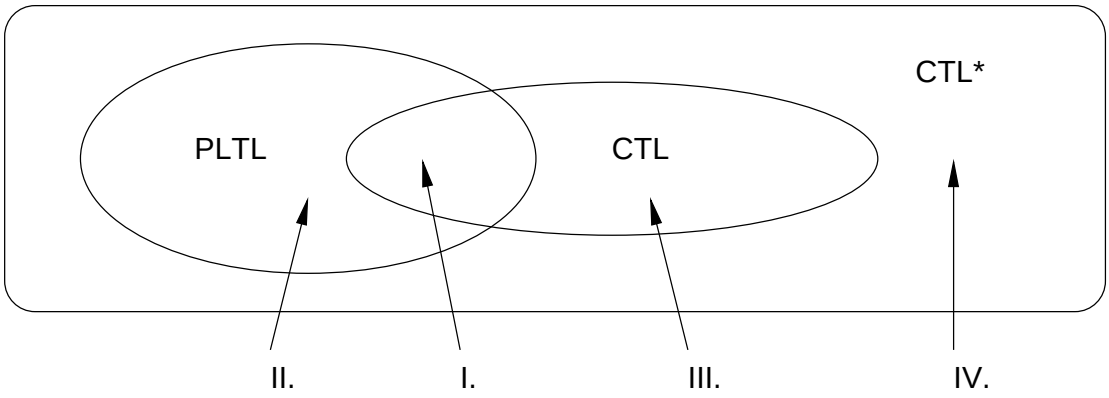
A PLTL, CTL, és CTL* logikák használatát az eltérő kifejezőképességük, valamint a logikai kifejezések igazsága ellenőrzésének eltérő komplexitása indokolja. A továbbiakban a kifejezőképességet vizsgáljuk meg, az ellenőrzés komplexitására később térünk ki.

5.6.1. Kifejezőképesség

Egy logika kifejezőképességét akkor mondjuk nagyobbnak egy másik logikáénál, ha képes olyan tulajdonságok megfogalmazására, amire a másik nem.

Megállapították, hogy a lineáris idejű temporális logikák és az elágazó idejű temporális logikák (mint logika osztályok) kifejezőképessége nem összehasonlítható. Ez azt jelenti, hogy vannak olyan tulajdonságok, amik egy adott lineáris idejű temporális logikában leírhatók, de egy bizonyos elágazó idejű temporális logikában nem, és viszont. Az általunk tárgyalt három speciális logika kapcsolata a 6 ábrán foglalható össze. Tehát a PLTL és a CTL kifejezőképessége nem összehasonlítható, a CTL* kifejezőképessége viszont nagyobb, mint az előző kettőé. Az egyes tartományokban példaként olyan formulák szerepelnek, amik alapján a kifejezőképességben különbséget lehet tenni:

- I: $A(p\,U\,q)$ (ha PLTL kifejezéseket állapotokon akarunk kiértékelni, akkor implicit univerzális kvantort alkalmazunk).
- II: $A(F(p \wedge Xp))$.
- III: $AG(EF\,p)$.
- IV: $A(F(p \wedge Xp)) \vee AG(EF\,p)$.



6. ábra. A PLTL, CTL és CTL* logikák kapcsolata

5.6.2. Méltányosság (FairCTL)

Gyakorlati rendszerek ellenőrzésekor gyakran van szükség bizonyos mellékfeltételek szem előtt tartására. Például ha egy rendszer egy Reset jellel bármikor a kezdőállapotba vihető, akkor (a kezdeti vizsgálatok után) elhanyagolhatjuk azt az esetet, amikor a rendszer a Reset jel állandó aktív szintje miatt soha nem tud a kezdőállapotból kimozdulni. Egy másik példa: egy kétprocesszes rendszerben a vizsgálatok

során eltekintünk attól a helyzettől, amikor az egyik processz el sem indulhat. A *méltányosság* betartása itt tehát azt jelenti, hogy a (temporális logikai kifejezéssel) megadott tulajdonságot csak bizonyos mellékfeltétel(ek) fennállása esetén fogjuk kiértékelni.

A CTL egyik hátrányaként említhető meg, hogy nem alkalmas a méltányosság kifejezésére. A vizsgálatok során figyelembe veendő feltételeket útvonal-kifejezésekkel adhatnánk meg, de az útvonal-kifejezések használatát a CTL erősen korlátozza. Szükség van tehát egy olyan bővítésre, ami megengedi a méltányosság kifejezéséhez szükséges útvonal-kifejezések használatát (de még nem eredményez egy olyan bonyolult logikát, mint a CTL*). Ez a bővítés a FairCTL.

A FairCTL logikában a tulajdonság mint egy (p, ϕ) páros tekinthető, ahol

- ϕ a méltányosság kifejezésére szolgáló útvonal-kifejezés, ami a $GF\,p$ (másként $F^{\infty}p$) és $FG\,p$ (másként $G^{\infty}p$) PLTL temporális operátorokat tartalmazhatja;
- p egy CTL állapot-kifejezés, ahol az útvonalakra vonatkozó kvantorok A_ϕ illetve E_ϕ alakban jelennek meg (A_ϕ jelentése "minden méltányos útvonalra", E_ϕ jelentése pedig "van olyan méltányos útvonal"); a többi kvantor (F, G, X, U) változatlan.

A p kifejezést az olyan útvonalakon kell kiértékelni, amelyekre a méltányosság kifejezése igaz. Így tehát egy $A_\phi F\,p$ FairCTL kifejezés a $A(\phi \Rightarrow F\,p)$ CTL* kifejezésnek felel meg. Hasonlóan $E_\phi G\,p$ kifejezés CTL* logikában mint $E(\phi \wedge Gp)$ írható fel.

Általános formában (nem a FairCTL esetén) a méltányosság az alábbi kategóriák szerinti kifejezéseket jelenthet:

- Feltétel nélküli méltányosság: Egy útvonalon a feltétel nélküli méltányosság teljesül a p kifejezésre, ha $GF\,p$ teljesül, azaz végtelen sokszor igaz a p kifejezés.
- Gyenge méltányosság: Egy útvonalon a gyenge méltányosság teljesül a p kifejezésre és a ϕ feltételre (kényszerre), ha $FG\,\phi \Rightarrow GF\,p$ igaz. Ez azt jelenti, hogy ha valamikor ϕ folyamatosan teljesül, akkor majd p is végtelen sokszor igaz lesz. Tipikus példa az $FG\,\mathrm{enabled}(a) \Rightarrow GF\,\mathrm{executed}(a)$ kifejezés, tehát ha valami engedélyezetté válik, akkor végtelen sokszor bekövetkezik.
- Erős méltányosság: Egy útvonalon az erős méltányosság teljesül a p kifejezésre és a ϕ feltételre (kényszerre), ha $GF\,\phi \Rightarrow GF\,p$ igaz. Ez azt jelenti, hogy ha ϕ végtelen sokszor teljesül (de nem feltétlenül folyamatosan), akkor majd p is végtelen sokszor igaz lesz.

5.7. A Hennessy-Milner logika

A következőkben egy olyan logikát ismerünk meg, ami LTS-eken értelmezett. A Hennessy-Milner logika (HML) egy egyszerű elágazó idejű logika, ami véges hosszúságú utakon képes tulajdonságokat megfogalmazni. Elsősorban a tesztelés során a tesztek, illetve a nem teljesülő tesztek esetén az ellenpéldák formalizálására használják.

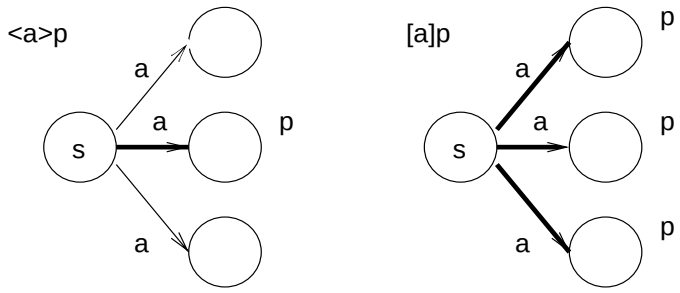
5.7.1. Formális szintaxis

A HML kifejezések a következő szabályok alapján állíthatók össze:

- (H1) true és false érvényes HML kifejezések.
- (H2) Ha p és q HML kifejezések, akkor $p \wedge q$ és $p \vee q$ is érvényes HML kifejezések.
- (H3) Ha p egy HML kifejezés és $a \in Act$ egy akció, akkor $[a]p$ egy érvényes HML kifejezés.

- (H4) Ha p egy HML kifejezés és $a \in Act$ egy akció, akkor $\langle a \rangle\, p$ egy érvényes HML kifejezés.

A H3 és H4 szabályok által bevezetett új operátorok egy adott állapotból az a akcióval elérhető következő állapotokra vonatkoznak. $[a]$ minden ilyen állapotra, $\langle a \rangle$ pedig egy létező ilyen állapotra írja elő, hogy p igaz legyen (ld. 7. ábra).



7. ábra. A Hennessy-Milner logika temporális operátorai

5.7.2. Formális szemantika

A HML kifejezések $T = (S, Act, \rightarrow)$ LTS-en értelmezettek. Az LTS egy $s \in S$ állapotára megadva:

- (H1) $T, s \models \mathrm{true}$, $T, s \not\models \mathrm{false}$ (true minden állapotban igaz, false egy állapotban sem igaz).
- (H2) $T, s \models p_1 \wedge p_2$ a.cs.a. $T, s \models p_1$ és $T, s \models p_2$.
  $T, s \models p_1 \vee p_2$ a.cs.a. $T, s \models p_1$ vagy $T, s \models p_2$.
- (H3) $T, s \models [a]\, p$ a.cs.a. $\forall s'$, ahol $s \stackrel{a}{\rightarrow} s' : T, s' \models p$.
- (H4) $T, s \models \langle a \rangle\ p$ a.cs.a. $\exists s' : s \stackrel{a}{\rightarrow} s'$ és $T, s' \models p$.

A Hennessy-Milner logika véges akciószekvenciák megadására alkalmas. Pl. egy adott állapotra $\langle a \rangle\, \mathrm{true}$ azt írja elő, hogy létezik a-val címkézett kimenő állapotátmenet; $[a]\mathrm{false}$ azt adja meg, hogy nincs a-val címkézett átmenet. $\langle a \rangle\, \langle b \rangle\, \langle c \rangle\, \mathrm{true}$ egy három akcióból álló akciósorozatot ír elő.

Ha a HML formulákat Kripke-struktúrákon szeretnénk értelmezni, akkor az eddigi true és false atomi kijelentések mellé be kell vezetni az AP atomi kijelentések halmazát: egy $P \in AP$ esetén $T, s \models P$ a.cs.a. $P \in L(s)$. Ugyanakkor a kvantorokat címkézetlen átmenetek esetén az [] alakban (bármely átmenet) illetve <> alakban (létezik átmenet) lehet megadnunk.

A fenti meggondolások alapján Kripke állapotátmeneti rendszereken (KTS) is értelmezhető a HML, ilyenkor mind az atomi kijelentések, mind az akciók (tehát az $[a]$ és $\langle a \rangle$ alakú operátorok) használhatók.

6. Modell ellenőrzés

Az előző fejezetben megismert temporális logikákat Kripke-struktúrákon (illetve LTS-eken) értelmeztük, a rendszer tulajdonságait ezeken a modelleken fogalmaztuk meg. Egy p temporális logikai kifejezés és egy M struktúra viszonyának vizsgálata többféleképpen is felvetődhet.

A rendszer verifikációja során azt vizsgáljuk, hogy ha adott egy véges állapotú struktúra, akkor ezen a struktúrán igaz-e a megadott kifejezés. Ez a *modell ellenőrzési* (angolul *model checking*) probléma: adott tehát egy M struktúra (ezen belül π útvonal illetve s állapot) és egy p kifejezés; fennáll-e, hogy

$M, \pi \models p$ (lineáris idejű temporális logika esetén) illetve $M, s \models p$ (elágazó idejű temporális logika esetén).

A modell ellenőrzés nem tévesztendő össze a klasszikus logika kielégíthetőség problémájával: ott az a kérdés, hogy van-e olyan struktúra, hogy a kifejezés igaz lesz. Egy p kifejezés *kielégíthető* a.cs.a. van olyan M struktúra, hogy $M \models p$.

Szokásos logikai probléma még az azonosan igaz kifejezések keresése. Egy kifejezés *azonosan igaz* (jelölése $\models p$) a.cs.a. minden struktúrán kielégíthető.

A továbbiakban a modell ellenőrzés problémájával foglalkozunk.

6.1. A modell ellenőrzés módszerei

A modell ellenőrzés két módját különböztethetjük meg:

- Globális modell ellenőrzés: Adott egy M struktúra és egy p temporális logikai kifejezés. Keressük az összes M-beli állapotot, amelyre p igaz. Eredményül tehát egy állapothalmazt kapunk.
- Lokális modell ellenőrzés: Adott egy M struktúra, egy p temporális logikai kifejezés és egy s állapot M-ben. Megvizsgáljuk, hogy M, s-re igaz-e p. Tipikusan a rendszer kiindulási állapotát szokás vizsgálni.

Látjuk, hogy a globális modell ellenőrzéssel a lokális modell ellenőrzés problémáját is megoldjuk, hiszen egy adott állapotra csak azt kell ellenőrizni, hogy eleme-e annak az állapothalmaznak, amit a globális modell ellenőrzés eredményeként kapunk.

6.1.1. A modell ellenőrzés technikái

A modell ellenőrzés következő technikái terjedtek el:

- Szemantikán alapuló megközelítés: Lényege, hogy egy kifejezés jelentését (vagyis azon állapotok halmazát, ahol a kifejezés igaz) a szemantikai szabályok felhasználásával, a struktúra alapján, induktívan számítjuk ki. Elsősorban globális modell ellenőrzésre alkalmas, elágazó logikák esetén.
- Automata-elméleti megközelítés: Automaták által elfogadott nyelvek vizsgálatára épül. A p kifejezésből egy A_p automatát hozunk létre, ami azokat az útvonalakat fogadja el, amelyekre p igaz. Egy másik automata, A_M az M struktúrából generálható, ez a struktúra által meghatározott útvonalakat fogadja el. Ezek után p igaz M-en a.cs.a. $L(A_M) \subseteq L(A_p)$. Ez a probléma redukálható az $A_M \times A_p^c$ szorzat-automata ürességének vizsgálatára. Az automata-elméleti megközelítés elsősorban lokális modell ellenőrzésre alkalmas, lineáris idejű logikák esetén.
- Tabló módszer. A lokális modell ellenőrzéshez egy bizonyítási fát próbálunk keresni, ami megmutatja, hogy az adott kifejezés igaz. A bizonyítási fa felépítése a struktúra alapján történik. Mind lineáris, mind elágazó idejű logikák modell ellenőrzésére alkalmas, de rendszerint bonyolult és nehezebben automatizálható, mint az előző két technika. Egyszerűbb logikák, mint például a HML esetén viszont jól használható.

A modell ellenőrzés során szükség lehet a temporális logikai kifejezések szintaktikai manipulációjára. Ehhez meg kell adnunk szabályoknak egy halmazát, amik egy adott kifejezésből szemantikailag ekvivalens kifejezést állítanak elő (tehát a kifejezések ugyanazokon az állapotokon igazak minden M esetén). Az ilyen szabályok összességét *axiomatizációnak* nevezik. Példaként megadunk néhány ilyen szabályt PLTL esetén (nem teljes szabálykészlet):

- Dualitás: $G\,p \equiv \neg F \neg p$, $F\,p \equiv \neg G \neg p$, $\neg X\,p \equiv X \neg p$
- Idempotencia: $GG\,p \equiv G\,p$, $FF\,p \equiv F\,p$, ...
- Abszorpció: $FGF\,p \equiv GF\,p$, $GFG\,p \equiv FG\,p$
- Kommutativitás: $X(p\,U\,q) \equiv (Xp)\,U\,(Xq)$
- Expanzió: $F\,p \equiv p \vee XF\,p$, $G\,p \equiv p \wedge XG\,p$

Ezeket a szabályokat alkalmazva kifejezéseket egyszerűsíthetünk (pl. modell ellenőrzés előtt), azonosan igaz kifejezéseket bizonyíthatunk illetve ekvivalens kifejezéseket találhatunk.

6.1.2. Az állapottér kezelése

A modell ellenőrzés problémája a struktúra vagyis az állapottér nagy mérete. Konkurens rendszerekben pl. gyorsan ”felrobban” a globális állapottér a független műveletek sokféle sorrendben lehetséges végrehajtása miatt. A probléma kezelésére két technika terjedt el:

- A bináris döntési diagramokat alkalmazó *szimbolikus technikát* tipikusan CTL kifejezések szemantikai alapú modell ellenőrzésére használják.
- A részleges rendezés (*partial ordering*) redukciót PLTL kifejezések automata-elméleti alapú modell ellenőrzésekor alkalmazzák.

A következő alfejezetekben áttekintjük az ellenőrzési technikákat és az állapottér hatékony kezelésének módszereit.

6.2. HML kifejezések modell ellenőrzése a tabló módszerrel

A bizonyításkeresés alapelveinek megértéséhez tekintsük először a Boole-logikát. A bizonyításkeresés a Boole-logikai kifejezések felbontásán alapul, és a gyakorlatban egy fa felépítését jelenti, ahol az egyes csomópontokban ”gyűjtjük” a felbontásnak megfelelő kifejezéseket. A felbontás szabályai a következők:

- VAGY jellegű operátorok esetén a fa két ágra bomlik: bármelyik ág igazzá válása elég az összetett kifejezés igazzá válásához.
- ÉS jellegű operátorok esetén a rész-kifejezéseket listába gyűjtjük; a lista minden kifejezése igaz kell legyen az összetett kifejezés igazzá válásához.

A felbontás előtt a kifejezést *negált normál formára* kell hozni, azaz a negálás operátorát azonosságok (pl. de Morgan szabályok) segítségével átvinni az összetett kifejezésekről az atomi kijelentésekre. Ezek után a felbontási szabályok a következők (felül az eredeti kifejezés, alul pedig a felbontott kifejezés(ek) találhatók; F az esetleg már listában lévő rész-kifejezéseket jelenti):

$$\frac{F, p \wedge q}{F, p, q}$$, azaz bővítjük a rész-kifejezések listáját.

$$\frac{F, p \vee q}{F, p} \quad \frac{F, p \vee q}{F, q}$$, azaz két ágra bomlik a fa.

$$\frac{F, p \Rightarrow q}{F, \neg p} \quad \frac{F, p \Rightarrow q}{F, q}$$, azaz itt is két ágra bomlik a fa.

A bontás addig történik, amíg atomi kijelentésekhez vagy azok negáltjaihoz nem jutunk. Példaként tekintsük a $P \vee Q \Rightarrow P \wedge Q$ kifejezésből építhető fát:

$$P \vee Q \Rightarrow P \wedge Q$$

$$\neg P \wedge \neg Q \qquad P \wedge Q$$

$$\neg P, \neg Q \qquad P,\ Q$$

A fa egy ága *ellentmondásos*, ha az ág végén (a levélben) egy kijelentés és annak negáltja is előfordul egy listában. A fa nem ellentmondásos (sikeres) ágai jelölik ki a kifejezés modelljét.

Nézzük meg, hogyan használható a tabló módszer a HML kifejezések modell ellenőrzésére! Azt kell tehát megmutatnunk, hogy $M, s \models p$. Ez a p kifejezés felbontásával történik, az s-ből elérhető állapottér felderítése alapján. A kérdést fogalmazzuk meg $s \vdash p$ alakban (M implicit), majd próbáljuk meg p-t felbontani rész-kifejezésekre. A bizonyításkeresés konstruálásának szabályai Hennessy-Milner logika esetén a következők:

$$\frac{s \vdash p_1 \wedge p_2}{s \vdash p_1\,, \quad s \vdash p_2}$$

$$\frac{s \vdash p_1 \vee p_2}{s \vdash p_1} \qquad \frac{s \vdash p_1 \vee p_2}{s \vdash p_2}$$

$$\frac{s \vdash [a]p}{s_1 \vdash p,\ \ldots,\ s_n \vdash p} \text{ ha } \{s_1, s_2, ..., s_n\} = \{s' \mid s \xrightarrow{a} s'\}$$

$$\frac{s \vdash \langle a \rangle\, p}{s' \vdash p} \text{ ha } s \xrightarrow{a} s'$$

Az utolsó két szabályból látszik, hogy ténylegesen az állapottér alapján történik a fa felépítése, az s állapotból egy a akcióval közvetlenül elérhető állapotok határozzák meg a felbontást. Az $[a]p$ tulajdonság ÉS jellegű, hiszen minden a-val elérhető következő állapotról mondunk valamit, az $\langle a \rangle\, p$ tulajdonság pedig VAGY jellegű, hiszen a szabályt minden, a kritériumnak megfelelő s'-re alkalmazhatjuk. Az utolsó szabály alkalmazása során arra kell ügyelnünk, hogy kerüljük el az ismétléseket (körbenjárást az átmenetek mentén).

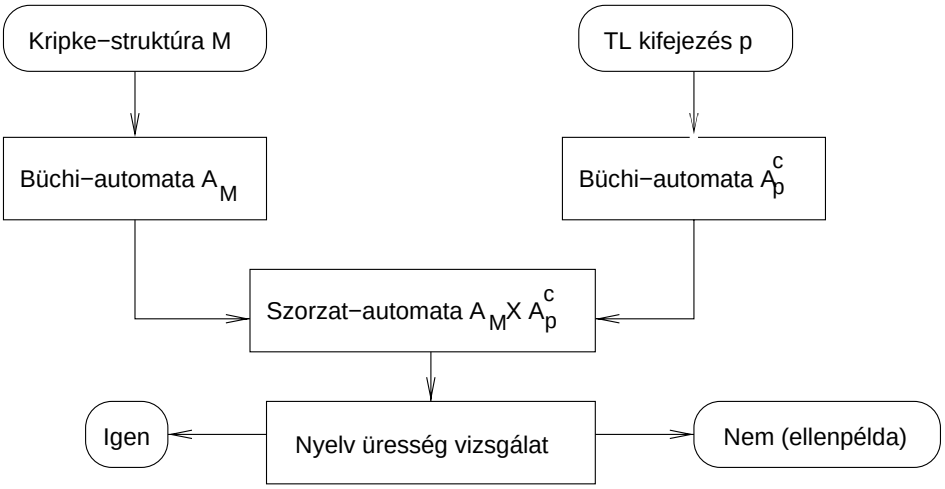
A tabló sikeres ágainak végén vagy $s \vdash \mathrm{true}$ alakú, vagy $s \vdash [a]p$ alakú formulákat (ahol nincs a átmenet s-ből) találunk. A tabló sikeres ágai jelölik ki a kifejezés modelljét.

6.3. PLTL modell ellenőrzés az automata-elméleti módszerrel

Egy p kifejezés az M Kripke-struktúrán történő modell ellenőrzése és az automata-elmélet a következőképpen hozhatók összefüggésbe:

- Egy s állapot egy $L(s)$ betűt azonosít a 2^{AP} ábécéből (a "betű" 2^{AP} egy részhalmaza, az adott állapothoz rendelt atomi kijelentések halmaza).
- Egy $\pi = (s_0, s_1, s_2, ...s_n)$ véges útvonal a $w_\pi = (L(s_0), L(s_1), ..., L(s_n))$ szóval azonosítható.
- Az $M = (S, R, L)$ Kripke-struktúra alapján konstruálható egy A_M automata, amely azokat (és csakis azokat) a szavakat fogadja el, amelyek megfelelnek M maximális véges útjainak.
- A p PLTL kifejezés alapján konstruálható egy A_p automata, amely azokat (és csakis azokat) a szavakat fogadja el, amelyek olyan utakhoz tartoznak, ahol p igaz.

Ezek után az $M \models p$ modell ellenőrzéshez azt kell megvizsgálnunk, hogy $L(A_M) \subseteq L(A_p)$, azaz a struktúrához tartozó nyelv része-e a tulajdonsághoz tartozó nyelvnek. Ez $L(A_M) \cap L(A_p)^c = \emptyset$ vizsgálatával is eldönthető. Ehhez először konstruálnunk kell az A_p^c komplementer automatát (teljesen definiált és determinisztikus esetben csak az elfogadó és nem elfogadó állapotokat kell felcserélni). A nyelvi metszetek üressége az $A_M \times A_p^c$ szinkron szorzat automata által elfogadott nyelv ürességének vizsgálatával ellenőrizhető (8. ábra). Ehhez azt kell megvizsgálni, hogy létezik-e a kezdőállapotból elérhető elfogadó állapot. Ha nem, akkor a nyelv üres, tehát teljesül a feltétel, a p kifejezés igaz az M Kripke-struktúrán.



8. ábra. Automata-elméleti megközelítés

Ha végtelen útvonalakat (végtelen hosszúságú szavakat) is kezelni szeretnénk, akkor a véges automaták helyett a Büchi-automaták osztályát kell használnunk. Ez valamelyest elbonyolítja az algoritmusokat (szorzat automata készítése, nyelv ürességének ellenőrzése), de az alapelv ugyanaz marad. A nyelvi üresség ellenőrzése itt az elfogadó futások létezésének ellenőrzését jelenti – ez végső soron cikluskeresésre vezethető vissza.

Az 9. ábra bal felső részén egy M Kripke-struktúrához tartozó A_M automata látható. A jobb oldalon a $p = F(P) \wedge G(Q)$ kifejezéshez tartozó A_p, alatta pedig az A_p^c automata látható. Az ábra alsó részén található a két automata szinkron szorzata. Mint látható, a szorzat automatában nincs elérhető elfogadó állapot, tehát a nyelv üres, igaz az M Kripke-struktúrán (1-es kezdőállapottal) a p kifejezés.

A továbbiakban az automaták mechanikus konstruálásának lépéseit tekintjük át.

6.3.1. Automata konstruálása a Kripke-struktúra alapján

Az $M = (S, R, L)$ Kripke-struktúrából a címkék áthelyezésével és elfogadó állapot létrehozásával konstruálható automata:

$$A_M = (2^{AP}, S \cup \{s_f\}, \{s_0\}, \rho, \{s_f\})$$

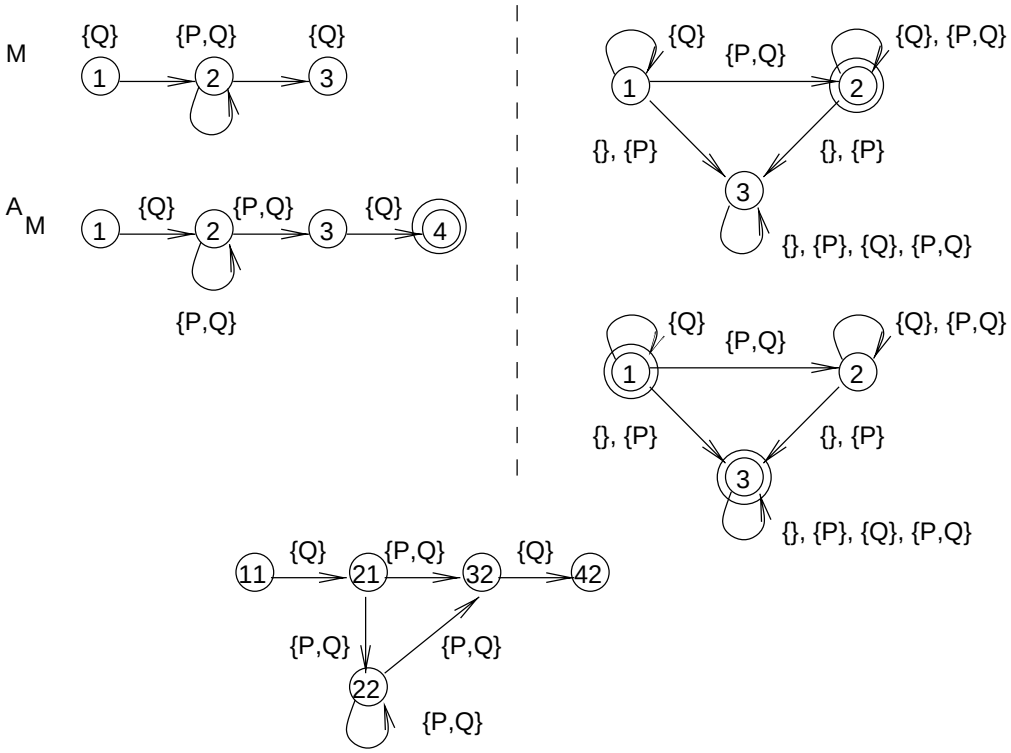
ahol s_f egy újonnan létrehozott állapot (az elfogadó állapot), és

$$\rho = \{(s, L(s), t) \mid (s, t) \in R\} \cup \{(s, L(s), s_f) \mid \text{nincs } t, \text{hogy } (s, t) \in R\}$$

Az automatát teljesen definiálttá és determinisztikussá kell tenni, majd célszerű minimalizálni.

6.3.2. Automata konstruálása PLTL kifejezésből

Most röviden vázoljuk fel az A_p automata konstruálásának alapötletét. A p PLTL kifejezést a tabló szabályai szerint bontjuk fel, és meghatározott esetekben a felbontási fa egyes csomópontjaiból egy M_p



9. ábra. Egy példa automaták szorzatára

Kripke-struktúra állapotait hozzuk létre. M_p lehetséges állapotsorozataihoz (útvonalaihoz) rendelt címkék adják meg az A_p által elfogadott nyelvet (M_p ismeretében A_p előállítható).

A felbontás során a tabló egy csomópontjának 3 mezőjét tároljuk: *New* tartalmazza a felbontandó kifejezéslistát, *Old* a már feldolgozott kifejezést (ami az adott állapotra vonatkozik), *Next* pedig a következő állapotra vonatkozó kifejezést (ami az adott állapotban feldolgozandó kifejezésből a következő állapotra utal, lásd pl. az X operátort).

Az automata állapotait az N listában gyűjtjük. Ebbe a listába akkor kerülhet egy csomópont a tablóból, ha (1) csak atomi kijelentések vagy ilyenek negáltjai szerepelnek a csomópont *New* mezőjének listájában, és (2) az N listában még nem szerepel olyan csomópont, aminek *Old* mezőjében ugyanilyen kijelentések vannak. Ha egy n csomópont N-be kerül, akkor a következőket kell tenni:

1. Mielőtt az N listába kerül egy csomópont, a *New* mezőben lévő listát az *Old* mezőbe másoljuk át.

2. Ha az n csomópont Next mezőjében van kifejezés, akkor létre kell hozni egy új m csomópontot, amelynek *New* mezőjébe kerül az n *Next* mezőjében lévő kifejezés. Ezután ezt az m csomópontot kell majd felbontani a tabló szabályai szerint.

3. Ha létrehoztunk egy új m csomópontot, akkor egy állapotátmenetet is létre kell hozni n-től m-be. Amikor az m-ből származó csomópont(ok) N-be kerül(nek), akkor ez(ek) ”örökli(k)” ezt az n-ből induló állapotátmenetet.

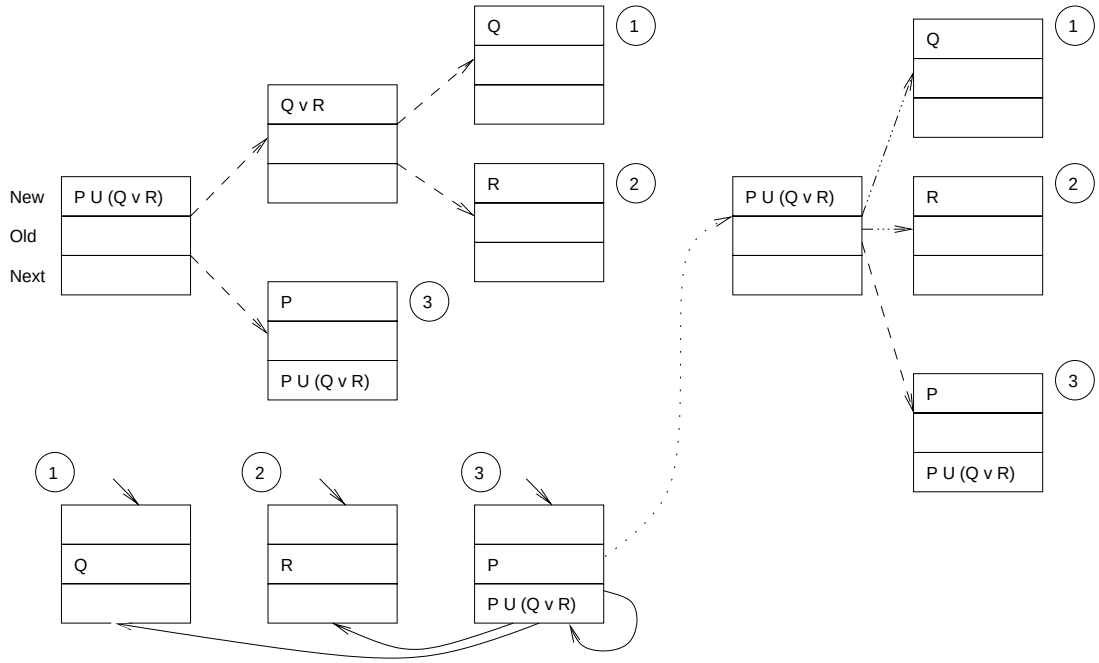
Így tehát a következő állapotra vonatkozó kifejezést ”áttesszük” az új csomópontba, amiből a következő állapot lehet a felbontás során.

A kezdeti p kifejezésből származó, N-be kerülő csomópontok lesznek az automata kezdeti állapotai.

A *New* mezőben lévő kifejezés felbontása (vagyis a tabló felépítése) a következőképpen történik PLTL esetén:

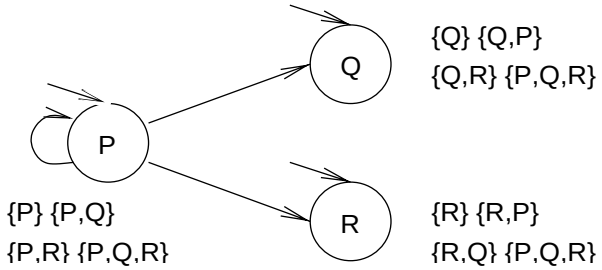
- $F, p \wedge q$ esetén a *New* mezőbe F, p, q kerül (lista elemek).
- $F, p \vee q$ esetén a csomópontot ketté kell osztani, a *New* mezőkben F, p illetve F, q kifejezésekkel.
- $F, X\, p$ esetén a *New* mezőben F marad, a *Next* mezőbe pedig p kerül (mivel ezt a következő állapotban kell majd vizsgálni, ha a csomópont vagy származékai N-be kerülnek).
- $F, p\, U\, q$ esetén a $p\, U\, q = q \vee (p \wedge X(p\, U\, q))$ azonosság alapján kell kettéosztani a csomópontot; az egyik csomópont *New* mezőjében F, q marad, a másik csomópont *New* mezőjébe F, p, a *Next* mezőjébe pedig $p\, U\, q$ kerül.

A p kifejezéshez tartozó automata állapotai tehát az N listában gyűjtött állapotok lesznek, a felbontás során a köztük lévő állapotátmeneteket is létrehozzuk. Az állapotokhoz tartozó címkéket úgy állapítjuk meg, hogy az AP atomi kijelentések halmazának az *Old* mezőben található atomi kijelentésekkel kompatibilis részhalmazait írjuk oda az egyes állapotokhoz (tehát ami tartalmazza az *Old* mezőben szereplő atomi kijelentéseket, de az ott szereplő negáltakhoz tartozókat nem). Ez azért történik így, mert ennek az automatának minden *megengedett* viselkedést le kell írnia.



10. ábra. Automata konstruálása egy PLTL kifejezésből

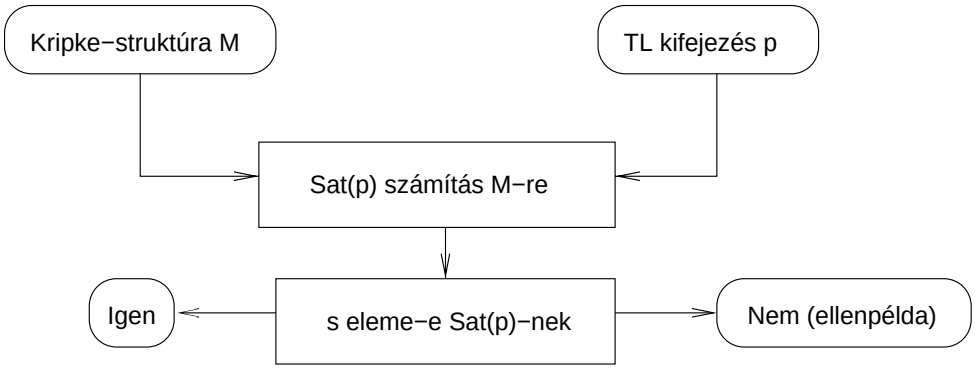
Példaként tekintsük a $p = P\, U\, (Q \vee R)$ kifejezés felbontását a 10. ábrán. Felül bontjuk ki az eredeti kifejezést, amiből az N állapotlista első három állapota lesz. Ezek közül a harmadik esetén találhatunk a *Next* mezőben értéket, tehát létre kell hoznunk egy állapotátmenetet (pontozott vonallal jelölve) és egy új csomópontot, majd azt kell felbontanunk (az ábra jobb oldala). Mivel ugyanazokat a csomópontokat kapjuk vissza, a pontozott vonallal jelölt állapotátmenet az eredeti 1, 2, 3 jelölésű csomópontokra ”öröklődik” (ezek az állapotátmenetek vannak folytonos vonallal rajzolva). A keletkezett automata a 11. ábrán látható.



11. ábra. A PLTL kifejezésből konstruált automata

6.4. CTL modell ellenőrzés a szemantikán alapuló módszerrel

A CTL modell ellenőrzési probléma annak az eldöntése, hogy egy adott $M = (S, R, L)$ Kripke-struktúra adott s_0 állapotára (kezdőállapot) és egy p CTL kifejezésre $M, s_0 \models p$ teljesül-e. Ez a globális modell ellenőrzés megközelítésében úgy történik, hogy kiszámítjuk azt az állapothalmazt, amelybe tartozó s állapotokra (mint potenciális kezdőállapotokra) a p kifejezés igaz. Legyen ennek az állapothalmaznak a jelölése $\mathrm{Sat}(p)$. Ezután megnézzük, hogy az adott s_0 benne van-e ebben a $\mathrm{Sat}(p)$ halmazban (12. ábra).



12. ábra. Szemantikán alapuló megközelítés

A $\mathrm{Sat}(p)$ állapothalmaz kiszámítása a CTL operátorainak formális szemantikája alapján történik, egy iterációs technikával. Ennek bevezetéséhez meg kell ismernünk az állapothalmazok közötti leképzésre vonatkozó matematikai tételeket.

6.4.1. Legkisebb és legnagyobb fixpontok teljes hálókban

Egy Kripke-struktúra S állapothalmazának 2^S hatványhalmaza (S összes részhalmaza) a $\subseteq$ részhalmaz relációra egy teljes hálót képez: a részhalmaz reláció parciális rendezés (reflexív, tranzitív, de nem szimmetrikus), a 2^S hatványhalmaz pedig tartalmazza a felső és alsó határt (a teljes S halmazt illetve az $\emptyset$ üres halmazt).

Legyen $\tau(z)$ egy állapothalmazok közötti, tehát egy $2^S \to 2^S$ leképzés. $\tau(z)$ tehát egy állapothalmazt egy másikra képez le, a z változó az állapothalmazokon értelmezett.

A $(2^S, \subseteq)$ hálóban τ-nak a következő tulajdonságai lehetnek:

- τ *monoton*, ha $z_1 \subseteq z_2$ következménye $\tau(z_1) \subseteq \tau(z_2)$, vagyis a leképzés megőrzi a részhalmaz relációt.

- τ $\cup$-*folytonos*, ha $z_1 \subseteq z_2 \subseteq z_3 \subseteq ...$ következménye $\tau(\cup_i z_i) = \cup_i \tau(z_i)$, vagyis a $\cup$ művelet és a leképzés művelete felcserélhető.

- τ $\cap$-folytonos, ha $z_1 \supseteq z_2 \supseteq z_3 \supseteq ...$ következménye $\tau(\cap_i z_i) = \cap_i \tau(z_i)$, vagyis a $\cap$ művelet és a leképzés művelete felcserélhető.

Ha S véges, akkor a $\cup$- és $\cap$-folytonosságok következnek a monotonitásból.
Definiáljuk ezután a $\tau(z)$ leképzésnek a legkisebb és legnagyobb fixpontját a következőképpen:

- A legkisebb fixpont, jelölése $\mathrm{lfp}\,\tau(z)$, az a *legkisebb* $z \in 2^S$ halmaz, melyet újra megkapunk, ha a leképzést elvégezzük, azaz $z = \tau(z)$.

- A legnagyobb fixpont, jelölése $\mathrm{gfp}\,\tau(z)$, az a *legnagyobb* $z \in 2^S$ halmaz, melyet újra megkapunk, ha a leképzést elvégezzük, azaz $z = \tau(z)$.

Jelöljük τ^i-vel τ egymás utáni i számú alkalmazását. A legkisebb és legnagyobb fixpont számításához a következő tételek mondhatók ki:

- Tarski tétele: Ha S véges és τ monoton, akkor *létezik* τ-nak legkisebb fixpontja illetve legnagyobb fixpontja, ahol is
$$\mathrm{lfp}\,\tau(z) = \cap\{z \mid \tau(z) \subseteq z\}$$
$$\mathrm{gfp}\,\tau(z) = \cup\{z \mid \tau(z) \supseteq z\}$$

- Kleene tétele: Ha τ $\cup$-folytonos, akkor $\mathrm{lfp}\,\tau(z) = \cup_i \tau^i(\emptyset)$; ha τ $\cap$-folytonos, akkor $\mathrm{gfp}\,\tau(z) = \cap_i \tau^i(S)$.

  Véges S és monoton τ esetén létezik olyan i_0 egész, hogy
$$\mathrm{lfp}\,\tau(z) = \tau^{i_0}(\emptyset)$$
  Hasonlóan létezik olyan j_0, hogy
$$\mathrm{gfp}\,\tau(z) = \tau^{j_0}(S)$$
  mivel bizonyítható, hogy minden i, j-re $\tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset)$ illetve $\tau^j(S) \supseteq \tau^{j+1}(S)$.

Ezek alapján a legkisebb fixpont a $\tau^i(\emptyset)$, a legnagyobb fixpont a $\tau^i(S)$ iterációval számítható. Az iteráció megállási feltétele, hogy az így számított halmaz nem változik. Az iterációk számára korlátot ad az S halmaz mérete.

6.4.2. CTL operátorok fixpont karakterisztikái

Mire tudjuk használni az állapothalmazok közötti leképzések legkisebb illetve legnagyobb fixpontjának itt megismert számítását a CTL modell ellenőrzés során? Nagyon pongyolán fogalmazva a következőket mondhatjuk:

- A legkisebb fixpont iteráció az eshetőségek, tehát adott tulajdonságot teljesítő elérhető állapotok meghatározásakor használható. Ez hasznos lesz az F operátort tartalmazó, tehát EF, AF jellegű kifejezések vizsgálatakor.

- A legnagyobb fixpont iteráció a folyamatosan teljesülő igazságok, tehát adott tulajdonságot folyamatosan teljesítő utak meghatározásakor használható. Ez a G operátort tartalmazó, tehát EG, AG jellegű kifejezések vizsgálatakor lesz hasznos.

Az egyszerűség kedvéért az $EF\,p$, $EG\,p$ és $E(p_1 U p_2)$ operátorokat vizsgáljuk meg, a többieket pedig ezek segítségével írjuk majd fel. Ismét tételeket kell kimondanunk:

- $\mathrm{Sat}(EF\,p) = \mathrm{lfp}\,\tau(z)$ ahol $\tau(z) = \mathrm{Sat}(p) \cup \mathrm{pre}(z)$, és $\mathrm{pre}(z)$ jelöli azon állapotok halmazát, ahonnan létezik átmenet z-beli állapotba: $\mathrm{pre}(z) = \{s \mid \exists t : (s,t) \in R \wedge t \in z\}$.

  Mit is jelent ez? Ha ismert egy z állapothalmaz, akkor ennek segítségével kiszámíthatjuk azt a másik állapothalmazt, ahol a következő teljesül: vagy p igaz, vagy van olyan rákövetkező állapot, ami z-ben van. Ez jelenti most számunkra a $\tau(z)$ leképzést. Ha ennek a $\tau(z)$-nek a legkisebb fixpontját kiszámítjuk, akkor azokat az állapotokat kapjuk, ahol $EF\,p$ teljesül.

  A fenti állítást nem bizonyítjuk, de a következőkben megpróbáljuk a mögöttes meggondolást intuitívan áttekinteni. Írjuk fel a következő azonosságot: $EF\,p = p \vee EX\,EF\,p$, azaz $EF\,p$ akkor igaz, ha vagy p igaz, vagy van olyan rákövetkező állapot, ahol $EF\,p$ igaz. Egyészt tehát $\mathrm{Sat}(EF\,p)$ számítása $\mathrm{Sat}(p)$ és $\mathrm{Sat}(EX\,EF\,p) = \mathrm{pre}(\mathrm{Sat}(EF\,p))$ alapján történhet, a logikai $\vee$ műveletnek a halmaz unió felel meg. Másrészt az egyenlőség jobb oldalán ismét megjelenik az eredetileg vizsgált $EF\,p$ kifejezés, de már a következő állapotra vonatkoztatva. Itt jön tehát be az az iteráció, amit a fenti képletben z jelenít meg.

  A tétel bizonyításához be kell látnunk, hogy $\tau(z)$ monoton, $\mathrm{Sat}(EF\,p)$ fixpontja $\tau(z)$-nek, majd $\mathrm{Sat}(EF\,p)$ a legkisebb fixpontja $\tau(z)$-nek.

- $\mathrm{Sat}(EG\,p) = \mathrm{gfp}\,\tau(z)$ ahol $\tau(z) = \mathrm{Sat}(p) \cap \mathrm{pre}(z)$.

  A $\tau(z)$ leképzés itt a következő formában jelenik meg: Ha ismert egy z állapothalmaz, akkor ennek segítségével kiszámíthatjuk azt a másik állapothalmazt, ahol a következő teljesül: p igaz, és van olyan rákövetkező állapot, ami z-ben van. Ha ennek a $\tau(z)$-nek a legnagyobb fixpontját kiszámítjuk, akkor azokat az állapotokat kapjuk, ahol $EG\ p$ teljesül.

  Az intuitív meggondolás alapja a következő azonosság: $EG\,p = p \wedge EX\,EG\,p$, azaz $EG\ p$ akkor igaz, ha p igaz, és van olyan rákövetkező állapot, ahol $EG\,p$ igaz. Tehát ismét megjelenik az eredetileg vizsgált kifejezés p mellett, de már a következő állapotra vonatkoztatva. Itt jön tehát be az az iteráció, amit a fenti képletben z jeleníti meg.

- $\mathrm{Sat}(E(p_1\,U\ p_2)) = \mathrm{lfp}\,\tau(z)$ ahol $\tau(z) = \mathrm{Sat}(p_2) \cup (\mathrm{Sat}(p_1) \cap \mathrm{pre}(z))$.

  A $\tau(z)$ leképzés itt a következő formában jelenik meg: Ha ismert egy z állapothalmaz, akkor ennek segítségével kiszámíthatjuk azt az állapothalmazt, ahol a következő teljesül: p_2 igaz, vagy p_1 igaz és van olyan rákövetkező állapot, ami z-ben van. Ez jelenti most számunkra a $\tau(z)$ leképzést. Ha ennek a $\tau(z)$-nek a legkisebb fixpontját kiszámítjuk, akkor azokat az állapotokat kapjuk, ahol $E(p_1\,U\ p_2)$ teljesül.

  Az intuitív meggondolás alapja a következő azonosság: $E(p_1\,U\ p_2) = p_2 \vee (p_1 \wedge EX\,E(p_1\,U\ p_2)$, azaz $E(p_1\,U\ p_2)$ akkor igaz, ha vagy p_2 igaz, vagy p_1 igaz és van olyan rákövetkező állapot, ahol $E(p_1\,U\ p_2)$ igaz. Ismét megjelenik itt is az eredetileg vizsgált kifejezés, de már a következő állapotra vonatkoztatva. Az iterációt a fenti képletben z jelenít meg.

CTL kifejezések és véges S állapothalmaz esetén a $\tau(z)$ leképzés monoton, így $\cup$- és $\cap$-folytonos is, tehát a fixpontok Kleene tétele alapján számíthatók.

6.4.3. Modell ellenőrzés iteratív módszerrel

Ezek után már meg is fogalmazhatjuk egy p CTL kifejezés modell ellenőrzésének módszerét.

1. A legnagyobb fixpont esetén a teljes állapothalmazt, a legkisebb fixpont esetén pedig az üres állapothalmazt vesszük kiindulásként.
2. Kiválasztunk egy p' rész-kifejezést, amelyben nincs beágyazott (más temporális operátor hatókörében álló) CTL operátor. Iterációval meghatározzuk a szemantikáját, azaz azoknak az állapotoknak a halmazát, ahol igaz. Ezt egy $A_{p'}$ atomi kijelentéssel "jelöljük meg", azaz így helyettesítjük a kifejezésben.
3. Az előző pontot ismételjük, míg teljesen fel nem dolgoztuk a kifejezést.

Eredményként egy p kifejezéshez egy $\mathrm{Sat}(p)$ állapothalmazt kapunk. Ezek (mint lehetséges kezdőállapotok) a globális modell ellenőrzési probléma megoldását adják.

A $\mathrm{Sat}(p)$ állapothalmaz felépítése tehát rekurzívan történik. A formális szintaxis ismertetésekor megadott összeállítási szabályokat mintegy "visszafelé" alkalmazva p egyszerűbb kifejezésekre bontható; a felbontásnak megfelelően pedig $\mathrm{Sat}(p)$ is összeállítható az egyes operátoroknak megfelelően az alábbiak szerint:

- $\mathrm{Sat}(P) = \{s \mid P \in L(s)\}$, ahol P egy atomi kijelentés, L pedig az állapotok címkézése volt. Itt $\mathrm{Sat}(\mathrm{true}) = S$, illetve $\mathrm{Sat}(\mathrm{false}) = \emptyset$.
- $\mathrm{Sat}(\neg p) = S \setminus \mathrm{Sat}(p)$, ahol p egy tetszőleges CTL kifejezés; hasonlóan
  $\mathrm{Sat}(p \wedge q) = \mathrm{Sat}(p) \cap \mathrm{Sat}(q)$,
  $\mathrm{Sat}(p \vee q) = \mathrm{Sat}(p) \cup \mathrm{Sat}(q)$.
- $\mathrm{Sat}(EX\, p) = \{s \mid \exists t : (s,t) \in R \wedge t \in \mathrm{Sat}(p)\}$, ez azt jelenti, hogy az állapotok közötti relációt kell figyelembe venni, és azokat az állapotokat "kikeresni", ahonnan elérhető $\mathrm{Sat}(p)$-beli állapot. Tulajdonképpen, az előző alfejezet jelölését használva, $\mathrm{Sat}(EX\, p) = \mathrm{pre}(\mathrm{Sat}(p))$.
- $\mathrm{Sat}(EF\, p)$ az előző alfejezet szerint legkisebb fixpont keresésével számítható:

  $$\mathrm{Sat}(EF\, p) = \mathrm{lfp}\, \tau(z) \text{ ahol } \tau(z) = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s,t) \in R \wedge t \in z\}$$

  Az $\mathrm{lfp}\, \tau(z)$ számítása során az iterációt az üres állapothalmazzal kell indítanunk, és alkalmaznuk kell Kleene tételét:

  $$\begin{aligned} z_0 &= \emptyset \\ z_1 &= \tau(z_0) \\ z_2 &= \tau(z_1) \end{aligned}$$

  és így tovább $z_{i+1} = \tau(z_i)$ egészen addig, míg $z_{i+1} = z_i$ nem lesz. Ekkor azt mondhatjuk, hogy megtaláltuk a legkisebb fixpontot, vagyis $\mathrm{Sat}(EF\, p)$-t.
  Általánosan felírva tehát az üres halmazból kiindulva a következő halmazokat a

  $$z_{i+1} = \tau(z_i) = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s,t) \in R \wedge t \in z_i\}$$

  képlettel kell számítanunk, amíg ugyanazt a halmazt vissza nem kapjuk az iteráció eredményeként.
- $\mathrm{Sat}(EG\, p)$ hasonlóan számítható, de itt a

  $$\mathrm{Sat}(EG\, p) = \mathrm{gfp}\, \tau(z) \text{ ahol } \tau(z) = \mathrm{Sat}(p) \cap \{s \mid \exists t : (s,t) \in R \wedge t \in z\}$$

  képlet alapján. Itt tehát a teljes állapothalmazból kiindulva a következő halmazokat a

  $$z_{i+1} = \tau(z_i) = \mathrm{Sat}(p) \cap \{s \mid \exists t : (s,t) \in R \wedge t \in z_i\}$$

  képlettel kell számítanunk, amíg ugyanazt a halmazt vissza nem kapjuk az iteráció eredményeként.

- $\mathrm{Sat}(E(p\,U\,q))$ számítása szintén iterációval történik, a

  $$\mathrm{Sat}(E(p_1\,U\;p_2)) = \mathrm{lfp}\,\tau(z) \text{ ahol } \tau(z) = \mathrm{Sat}(p_2) \cup (\mathrm{Sat}(p_1) \cap \{s \mid \exists t : (s,t) \in R \wedge t \in z\})$$

  képlet alapján. Az üres halmazból indulva az iteráció lépése a következőképpen írható fel:

  $$z_{i+1} = \tau(z_i) = \mathrm{Sat}(p_2) \cup (\mathrm{Sat}(p_1) \cap \{s \mid \exists t : (s,t) \in R \wedge t \in z_i\})$$

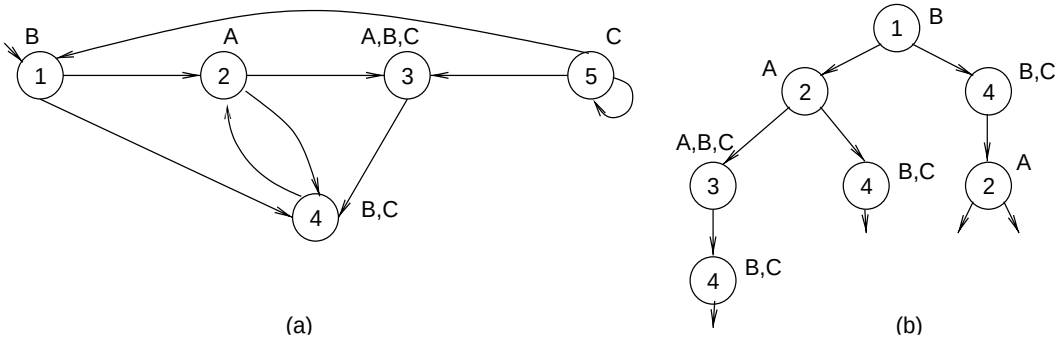
- A további CTL operátorokat már csak mint jelöléseket (rövidítéseket) tekinthetjük, amelyek a már ismert operátorok alapján számíthatók:

  $$\begin{aligned} AX\,p &= \neg\, EX(\neg p) \\ AF\,p &= \neg\, EG(\neg p) \\ AG\,p &= \neg\, EF(\neg p) \\ A(p\,U\,q) &= \neg\, E(\neg q\,U\,\neg p \wedge \neg q) \wedge \neg\, EG\,\neg q \end{aligned}$$

Ezek alapján már képesek vagyunk tetszőleges p CTL kifejezéshez tartozó $\mathrm{Sat}(p)$ halmaz kiszámítására, ezután az aktuális kezdőállapot tartalmazásának vizsgálata már egyszerű dolog.

6.4.4. Egy példa halmazokkal számolva

Adott a 13(a) ábrán látható véges állapotú rendszer. Az egyszerűség kedvéért az állapotokat az $S = \{1,2,3,4,5\}$ egész számokkal jelöltük, az állapotok címkézése pedig az $AP = \{A,B,C\}$ kijelentésekkel történt. A rendszer kezdőállapota az 1-gyel jelölt állapot. Azt kell ellenőriznünk, hogy ezen a struktúrán az $AG\,(A \vee C)$ CTL kifejezés igaz-e.



13. ábra. Egy véges állapotú rendszer (a) és a hozzá tartozó számítási fa (b)

Az 13(a) ábra szerinti rendszernek a 13(b) ábrán látható számítási fa felel meg – formálisan ezen értelmezett az ellenőrizendő kifejezés.

Az előző alfejezetben leírtak alapján $AG\,p = \neg\, EF\neg p$ és $\mathrm{Sat}(EF\,p)$ számítását a

$$z_{i+1} = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s,t) \in R \wedge t \in z_i\}$$

iterációval végezhetjük az üres halmazból kiindulva. Bontsuk fel tehát az ellenőrizendő kifejezést és határozzuk meg a megfelelő halmazokat:

- $\mathrm{Sat}(A) = \{2,3\}$, azaz az A-val címkézett állapotok halmaza.

- $\mathrm{Sat}(C) = \{3, 4, 5\}$, azaz a C-vel címkézett állapotok halmaza.
- $\mathrm{Sat}(A \vee C) = \mathrm{Sat}(A) \cup \mathrm{Sat}(C) = \{2, 3\} \cup \{3, 4, 5\} = \{2, 3, 4, 5\}$
- $\mathrm{Sat}(\neg(A \vee C)) = S \setminus \mathrm{Sat}(A \vee C) = \{1, 2, 3, 4, 5\} \setminus \{2, 3, 4, 5\} = \{1\}$.

Jelöljük a továbbiakban p-vel az $\neg(A \vee C)$ kifejezést, így $\mathrm{Sat}(p) = \{1\}$. Hozzáláthatunk az iterációval $\mathrm{Sat}(EF\, p)$ számításához:

- $z_0 = \emptyset$
- $z_1 = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s, t) \in R \wedge t \in z_0\} = \{1\} \cup \emptyset = \{1\}$
- $z_2 = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s, t) \in R \wedge t \in z_1\} = \{1\} \cup \{5\} = \{1, 5\}$
- $z_3 = \mathrm{Sat}(p) \cup \{s \mid \exists t : (s, t) \in R \wedge t \in z_2\} = \{1\} \cup \{5\} = \{1, 5\}$ azaz véget ért az iteráció, $\mathrm{Sat}(EF\, p) = \{1, 5\}$.

Ezután már a végeredményt kapjuk:

$$\mathrm{Sat}(AG(A \vee C)\,) = \mathrm{Sat}(\neg\, EF\, p) = S \setminus \mathrm{Sat}(EF\, p) = \{1, 2, 3, 4, 5\} \setminus \{1, 5\} = \{2, 3, 4\}$$

Mivel a kezdőállapot az 1-gyel jelölt állapot volt, ami nem eleme ennek a halmaznak, az $AG(A \vee C)$ kifejezés nem igaz ezen a struktúrán. (Itt egyszerű ellenpéldát keresni, hiszen már a kezdőállapot sem elégíti ki az $A \vee C$ kifejezést.)

6.5. Az állapottér kezelése szimbolikus technikával

Az előző alfejezetben tárgyalt iteratív technikának (önmagában) az a hátránya, hogy a nagy állapotterű rendszerek esetén természetesen az iteráció során is nagy méretű halmazokat kell tárolni, illetve rajtuk műveleteket végezni. A továbbiakban megpróbáljuk ezeket az állapothalmazokat kompaktabb formában reprezentálni.

6.5.1. Karakterisztikus függvények

A nagy méretű állapothalmazok tárolási és kezelési problémájának megoldását a szimbolikus technikák jelenthetik. Alapgondolatuk az, hogy a halmazok explicit tárolása helyett azok *karakterisztikus függvényét* tároljuk, és a halmazműveleteket is a karakterisztikus függvények segítségével végezzük el. A karakterisztikus függvények Boole-függvények lesznek, így az szükséges, hogy az állapotokat binárisan kódoljuk, azaz bitvektorokként írjuk fel. Az állapotváltozók számát a kódolandó állapotok száma határozza meg. Egy állapothalmaz karakterisztikus függvénye az az állapotváltozókon értelmezett logikai függvény, ami akkor és csakis akkor igaz, ha az állapot az adott halmazba tartozik.

A karakterisztikus függvények használatának matematikai alapját a Stone-tétel (Boole-algebrák reprezentációs tétele) adja: Minden véges Boole-algebra izomorf egy véges S halmaz részhalmazainak algebrájával. Itt

- Az S halmaz részhalmazainak algebrája $(2^S, \cup, \cap, \emptyset, S)$.
- A megfelelő Boole-algebra az n-változós logikai függvények algebrája: $(F_n, \vee, \wedge, 0, 1)$. Itt F_n a $\{0, 1\}$ felett értelmezett n-változós logikai függvények halmaza. Mivel 2^{2^n} ilyen függvény van, és $2^{|S|}$ számú részhalmaz lehetséges, ezért $|S| \leq 2^n$ összefüggés adja meg a szükséges állapotváltozók számát.

Legyen a továbbiakban az állapotváltozók halmaza $x_1, x_2, ..., x_n$. A karakterisztikus függvényeket a következőképpen definiáljuk:

- Egy s állapot kódolása legyen $x_1 = u_1, x_2 = u_2, ..., x_n = u_n$. Ekkor az s azaz $(u_1, u_2, ..., u_n)$ állapot karakterisztikus függvénye az a $C_s(x_1, x_2, ..., x_n)$ függvény, amely akkor és csakis akkor 1 értékű, ha $x_1 = u_1, x_2 = u_2, ..., x_n = u_n$. Egy konjunkciót kell tehát felírnunk az állapotváltozók között, ahol x_i operandus szerepel ha $u_i = 1$ és $\neg x_i$ szerepel ha $u_i = 0$.

- Egy $Y \subseteq S$ állapothalmaz karakterisztikus függvénye az a $C_Y(x_1, x_2, ..., x_n)$ függvény, amely akkor és csakis akkor 1 értékű egy $(u_1, u_2, ..., u_n)$ behelyettesítésre, ha $(u_1, u_2, ..., u_n) \in Y$.

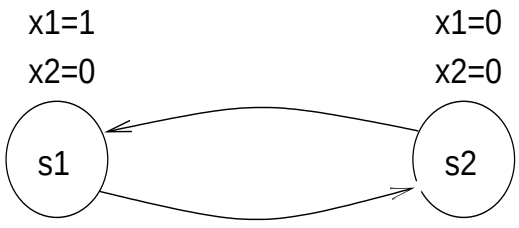
- Az állapothalmazok uniójának illetve metszetének karakterisztikus függvénye egyszerűen számítható:
  $C_{Y_1 \cup Y_2} = C_{Y_1} \vee C_{Y_2}$, illetve $C_{Y_1 \cap Y_2} = C_{Y_1} \wedge C_{Y_2}$. Így az állapothalmaz (mint az állapotok uniója) karakterisztikus függvénye is számítható az egyes állapotok karakterisztikus függvényeiből: $C_Y = \bigvee_{s_i \in Y} C_{s_i}$.

- Az állapotátmenetek karakterisztikus függvényét az állapotváltozók ”duplikálásával” fejezhetjük ki: Tekintsük az $r = (s, t) \in R$ átmenetet, ahol $s = (u_1, u_2, ..., u_n)$, $t = (v_1, v_2, ..., v_n)$. Azt kell kifejeznünk, hogy az átmenetbe s és t is ”beletartozik”, tehát első közelítésben $C_s \wedge C_t$ alakú kellene legyen a karakterisztikus függvény. Az átmenet kiindulási állapotát és a célállapotát úgy különböztetjük meg, hogy az utóbbit az $x'_1, x'_2, ..., x'_n$ (vesszős) állapotváltozókkal kódoljuk. Tehát az átmenet karakterisztikus függvénye

  $$C_r(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n)$$

  lesz, ami itt csak akkor 1 értékű, ha $x_1 = u_1, x_2 = u_2, ..., x_n = u_n$ valamint $x'_1 = v_1, x'_2 = v_2, ..., x'_n = v_n$. Vagyis $C_r = C_s \wedge C'_t$, ahol C'_t azt jelöli, hogy itt a vesszős állapotváltozókat használjuk.

- Az állapotátmenetek egy halmazának (mint az egyes átmenetek uniójának) karakterisztikus függvényét a már ismert unió formulával számíthatjuk ki: $C_R = \bigvee_{r_i \in R} C_{r_i}$.



14. ábra. Egy egyszerű rendszer bináris állapotváltozókkal

A 14. ábrán példaként egy kétállapotú rendszer látható az s_1 és s_2 állapotokkal, az x_1 és x_2 állapotváltozókkal kódolva. Az egyes állapotok illetve állapothalmazok karakterisztikus függvénye a következőképpen adható meg:
$C_{s_1} = x_1 \wedge \neg x_2$ (csak az $x_1 = 1$ és $x_2 = 0$ behelyettesítés esetén lesz 1 értékű).
$C_{s_2} = \neg x_1 \wedge \neg x_2$
$C_S = C_{\{s_1\} \cup \{s_2\}} = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_2)$
Az állapotátmeneti függvény az x_1, x_2 illetve x'_1, x'_2 változókkal adható meg:

$C_{(s_1,s_2)} = (x_1 \wedge \neg x_2) \wedge (\neg x'_1 \wedge \neg x'_2)$
$C_{(s_2,s_1)} = (\neg x_1 \wedge \neg x_2) \wedge (x'_1 \wedge \neg x'_2)$
$C_R = C_{\{(s_1,s_2),(s_2,s_1)\}} = C_{(s_1,s_2)} \vee C_{(s_2,s_1)}$

Az előző alfejezetekben láttuk, hogy a CTL modell ellenőrzés visszavezethető az állapothalmazokon végzett unió és metszet műveletekre. Így természetesen visszavezethető lesz a karakterisztikus függvényeken végzett $\vee$, $\wedge$ műveletekre is. Egyedül az $EX\,p$ alakú kifejezések esetén kell új műveletet bevezetni, mivel azt írtuk fel, hogy

$$\mathrm{Sat}(EX\,p) = \{s \mid \exists t : (s,t) \in R \wedge t \in \mathrm{Sat}(p)\},$$

azaz az R állapotátmeneti relációt kell úgy figyelembe venni, hogy megelőző állapotok halmazát kell "kikeresni" állapotok adott halmaza alapján. Itt $\mathrm{Sat}(p)$ az állapotok adott halmaza, ennek $C_{\mathrm{Sat}(p)}$ a karakterisztikus függvénye. A keresett állapotoknak tehát a $C_R \wedge C'_{\mathrm{Sat}(p)}$ függvényt kell igazzá tenniük.

Egy C függvény egzisztenciális absztrakcióját annak x változója szerint $\exists_x C$ alakban írjuk, ahol $\exists_x C = C[1/x] \vee C[0/x]$, itt pedig $C[1/x]$ jelöli C függvényt az $x = 1$ behelyettesítéssel, $C[0/x]$ pedig az $x = 0$ behelyettesítéssel.

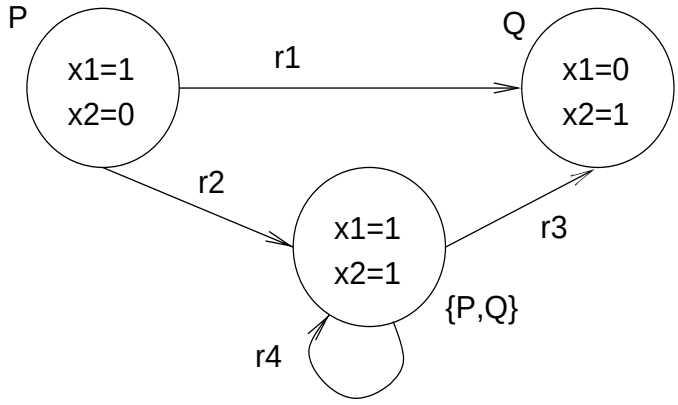
Tehát a fenti esetben az egzisztenciális absztrakció az $x'_1, x'_2, ..., x'_n$ változókra történik (a megelőző állapotokat keressük), tehát $C_{\mathrm{Sat}(EX\,p)} = \exists_{x'_1,x'_2,...,x'_n}(C_R \wedge C'_{\mathrm{Sat}(p)})$.

6.5.2. Egy példa karakterisztikus függvényekkel számolva

Példaként vegyük az 15 ábrán megadott rendszert. Az állapotváltozók x_1 és x_2, az atomi kijelentések P és Q. Az állapotátmenetek karakterisztikus függvénye a következő:

$C_R(x_1, x_2) = C_{r_1} \vee C_{r_2} \vee C_{r_3} \vee C_{r_4} = x_1 \wedge x'_2$

Legyen az ellenőrizendő CTL kifejezés a következő: $p = P \vee EF\,Q$.



15. ábra. Egy rendszer binárisan kódolt állapotokkal

A továbbiakban az egyszerűbb jelölés kedvéért azoknak az állapotoknak a karakterisztikus függvényét, ahol egy p kifejezés igaz, jelöljük C_p-vel $C_{\mathrm{Sat}(p)}$ helyett.

Számoljuk ki a karakterisztikus függvényeket:

$C_P = (x_1 \wedge \neg x_2) \vee (x_1 \wedge x_2) = x_1$, a P-vel címkézett állapotok "felsorolásával";

$C_Q = (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_2) = x_2$ hasonlóan.

Mint az előző alfejezetekből ismert, $\mathrm{Sat}(EF\,Q)$ számítása iterációval történik. Az iteráció kiindulása a $z_0 = \emptyset$, majd

$$z_{i+1} = \tau(z_i) = \mathrm{Sat}(Q) \cup \mathrm{Sat}(EX\,z_i)$$

volt. Ezt karakterisztikus függvényekkel elvégezve a következőket kapjuk:

- $C_{z_0} = \text{false}$
- $C_{z_1} = C_Q \vee C_{EX(C_{z_0})} = x_2 \vee C_{EX\,\text{false}} = x_2$, mivel azoknak az állapotoknak a karakterisztikus függvénye, ahol az EX false kifejezés igaz, természetesen false (nincs ilyen állapot).
- $C_{z_2} = C_Q \vee C_{EX(C_{z_1})} = x_2 \vee \exists_{x'_1,x'_2}(C_R \wedge C'_{z_1})$. Mivel C_R-ben x'_1 nem szerepel, valamint $C_R \wedge x'_2 = x_1 \wedge x'_2$, így folytathatjuk:
  $C_{z_2} = x_2 \vee \exists_{x'_2}(x_1 \wedge x'_2) = x_2 \vee ((x_1 \wedge x'_2)[1/x'_2] \vee (x_1 \wedge x'_2)[0/x'_2]) = x_2 \vee x_1$.
- $z_3 = C_Q \vee C_{EX(C_{z_2})}$.
  Itt $C_{EX(C_{z_2})} = \exists_{x'_1,x'_2}(C_R \wedge (x'_2 \vee x'_1)) = \exists_{x'_1,x'_2}((x_1 \wedge x'_2) \wedge (x'_2 \vee x'_1)) =$
  $\exists_{x'_1}\left(\exists_{x'_2}((x_1 \wedge x'_2) \wedge (x'_2 \vee x'_1))\right) =$
  $\exists_{x'_1}(((x_1 \wedge x'_2) \wedge (x'_2 \vee x'_1))[1/x'_2] \vee ((x_1 \wedge x'_2) \wedge (x'_2 \vee x'_1))[0/x'_2]) =$
  $\exists_{x'_1}((x_1 \wedge 1) \vee (0 \wedge x'_1)) = x_1$.
  Tehát $C_{z_3} = x_2 \vee x_1$.

Az iteráció véget ért, $C_{EF\,Q} = x_2 \vee x_1$.

Visszatérve az eredeti CTL kifejezésre: Azoknak az állapotoknak a karakterisztikus függvénye, ahol a $P \vee EF\,Q$ kifejezés igaz: $C_{P \vee EF\,Q} = C_P \vee C_{EF\,Q} = x_1 \vee (x_2 \vee x_1) = x_1 \vee x_2$. Ez a példában minden állapotra fennáll, tehát bármely állapotból indítva a rendszert, a kifejezés igaz.

6.5.3. BDD alapú reprezentáció

A karakterisztikus függvényeknek – mint Boole-függvényeknek – a tárolására és a műveletek elvégzésére a *redukált sorrendezett bináris döntési diagramok* (ROBDD) hatékonyan alkalmazhatók. Ezt vizsgáljuk meg a következőkben.

Vezessük be az úgynevezett *if-then-else* operátort, amit a következőképpen definiálhatunk:

$$x \to x_1, x_0 = (x \wedge x_1) \vee (\neg x \wedge x_0)$$

tehát a kifejezés x_1 értékét veszi fel, ha x igaz (true, 1), és x_0 értékét, ha x nem igaz (false, 0). x-et szokás teszt változónak is nevezni.

Egy x változón értelmezett f logikai függvény ez alapján felírható a következőképpen (Shannon-felbontás):

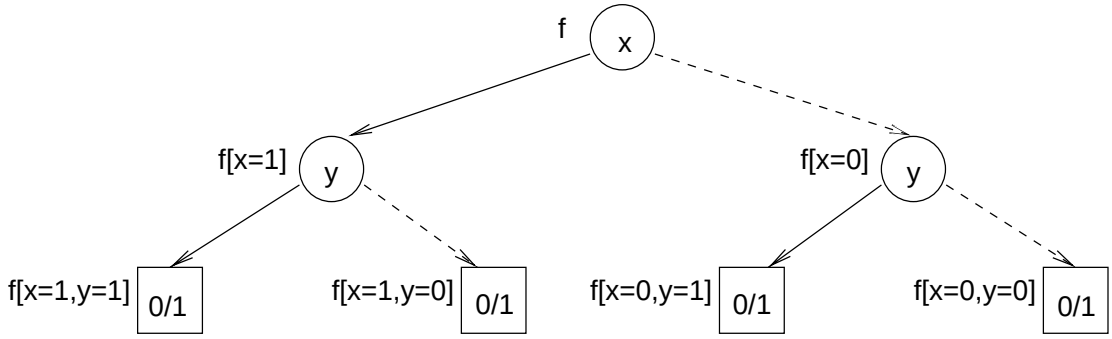
$$f = x \to f[1/x],\ f[0/x]$$

A továbbiakban az egyszerűség kedvéért f_x jelölje az $f[1/x]$ behelyettesítést és $f_{\overline{x}}$ az $f[0/x]$ behelyettesítést.

Vegyük észre, hogy a Shannon felbontásban a jobb oldal f_x és $f_{\overline{x}}$ tagjai eggyel kevesebb változót tartalmaznak (hiszen behelyettesítjük x értékeit). Természetesen a jobb oldalon szereplő kifejezéseket is tovább bonthatjuk (amíg van változó) az *if-then-else* operátor segítségével, újabb és újabb változókat kiválasztva és azokra felírva a helyettesítést. Így egy *bináris döntési fa* struktúrát kaphatunk, amelyben a csomópontok a logikai kifejezések, az ágak pedig az *if-then-else* operátorral való felbontások (behelyettesítések) alapján adódnak. Természetesen ha f összes változóját behelyettesítjük, akkor a 0 (false) vagy 1 (true) értékek egyikét kapjuk, ezek képezik a fa leveleit. Egy $f(x, y)$ Boole-függvény felbontása látható a 16 ábrán (0/1-gyel jelöltük, hogy 0 vagy 1 szerepel a felbontás végén).

Egy függvény helyettesítési értékét megkapjuk, ha a döntési fában a gyökértől indulva az egyes változók értékének megfelelő irányokban megyünk végig a fa ágain, míg a konstans 0 (false) vagy 1 (true) értéket reprezentáló levélhez nem jutunk. Ez lesz a függvény értéke.

A bináris döntési fát egyszerűsíthetjük a következőképpen:



16. ábra. Egy bináris döntési fa felépítése

- *Bináris döntési diagramot* (Binary Decision Diagram, BDD) kapunk, ha az azonos csomópontokat illetve részfákat összevonjuk.
- *Rendezett bináris döntési diagramot* (Ordered BDD) kapunk, ha a felbontás során minden ágon azonos sorrendben vesszük fel a teszt változókat.
- *Redukált rendezett bináris döntési diagramot* (Reduced Ordered BDD, ROBDD) kapunk, ha a diagramban a szükségtelen csomópontokat (amelyekből a felbontás során azonos csomóponthoz vezet mindkét ág) redukáljuk: a csomópontot töröljük, bemenő élét a kimenő élek által elért csomóponthoz irányítjuk.

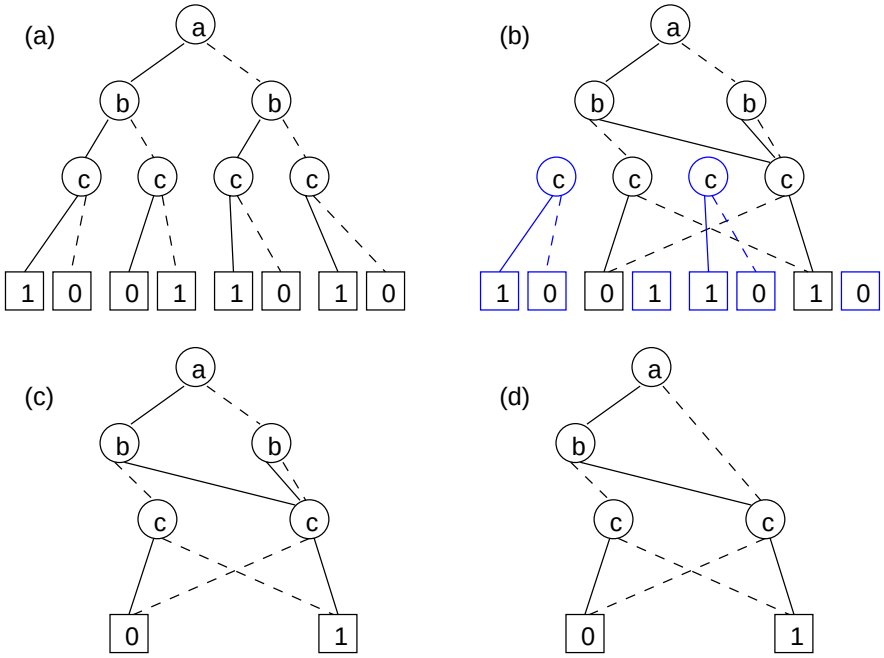
A 17(a) ábrán az $f = (b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge c)$ logikai függvényt reprezentáló döntési fát láthatjuk (a csomópontokba a teszt változókat írtuk bele). Az izomorf részfákat (illetve leveleket) összevonva a (b) lépésen keresztül kapjuk a BDD-t a (c) ábrán. Ez már rendezett is, mert azonos sorrendben szerepelnek a változók. A szükségtelen csomópontot redukálva a ROBDD alakot kapjuk a (d) ábrán.

A ROBDD reprezentáció, egy adott teszt változó sorrendezéssel, a Boole-függvényeknek egy kanonikus alakja, tehát két ROBDD, amely ugyanazt a Boole-függvényt reprezentálja azonos változó-sorrendezéssel, izomorf.

Összefoglalóul egy ROBDD a következő tulajdonságokkal rendelkezik:

- Irányított, aciklikus gráf, egy gyökérrel és két levéllel rendelkezik. A levelek a 0 (false) illetve az 1 (true). Minden csomópontot egy teszt változóval címkézhetünk.
- Minden csomópontból két kimenő él indul, ezek a 0 illetve 1 behelyettesítést reprezentálják.
- Minden útvonal mentén a teszt változók azonos sorrendben fordulnak elő.
- Az izomorf részfák egyetlen részfává vannak összevonva.
- Azok a csomópontok, ahonnan kimenő él ugyanahhoz a csomóponthoz vezet, redukálva vannak.

A ROBDD Boole-függvényeknek egy nagyon tömör reprezentációja lehet. A ROBDD tömörségére a jól ismert ”étkező filozófusok” példát említhetjük. Ebben a példában a filozófusok számától függően az állapottér igen nagy lehet, amit hagyományos (felsorolásos) módon már nem lehetne kezelni. Az alábbi táblázatban az állapotok száma illetve az adott méretű állapotteret leírni képes ROBDD mérete, azaz a a csomópontok száma szerepel. Mivel egy ROBDD csomópont kb. 16 bájton tárolható, az állapottér 28 filozófus esetén is csak kb. 21 kilobájtot foglal el a memóriában.



17. ábra.
Egy logikai függvény bináris döntési fa (a), redukált döntési fa (b), BDD (c) és ROBDD (d) reprezentációja

Filozófusok száma	Állapotok száma	ROBDD csomópontok száma
16	$4,7 * 10^{10}$	747
28	$4,8 * 10^{18}$	1347

Ugyanakkor tudnunk kell a következőket:

- A méret szempontjából igen lényeges a változók sorrendje: más sorrend (akár nagyságrendileg) eltérő számú ROBDD csomópontot eredményezhet. Az optimális sorrend sok esetben csak heurisztikus módszerekkel (pl. az egy processzhez tartozó állapotváltozókat egymás után felvéve) illetve ”próba-szerencse” alapon határozható meg.
- Ha egy konkurens rendszerben a komponensek állapotterei alapján konstruáljuk meg a globális állapotteret és eközben fokozatosan építjük a ROBDD-t, akkor az építés közben általában jóval több csomópontot kell tárolnunk, mint a végső, már teljes állapotteret reprezentáló ROBDD esetén. A fenti példában 28 filozófus esetén maximum 18734 csomópontra van szükség, a végére ez redukálódik 1347-re.

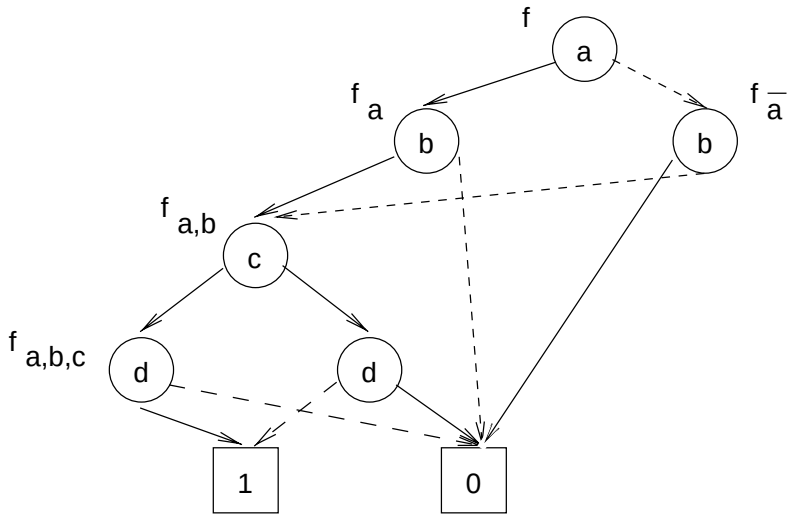
6.5.4. Példa egy ROBDD kézi előállítására

Tekintsük az $f = (a \Leftrightarrow b) \wedge (c \Leftrightarrow d)$ függvényt. A Shannon-felbontást az a, b, c, d változó-sorrendezéssel végezzük el.

- $f = a \rightarrow f_a, f_{\overline{a}} = a \rightarrow (1 \Leftrightarrow b) \wedge (c \Leftrightarrow d), (0 \Leftrightarrow b) \wedge (c \Leftrightarrow d)$
- $f_a = b \rightarrow f_{a,b}, f_{a,\overline{b}} = b \rightarrow (1 \Leftrightarrow 1) \wedge (c \Leftrightarrow d), (1 \Leftrightarrow 0) \wedge (c \Leftrightarrow d) = b \rightarrow (c \Leftrightarrow d), 0$

- $f_{\overline{a}} = b \to f_{\overline{a},b}, f_{\overline{a},\overline{b}} = b \to (0 \Leftrightarrow 1) \wedge (c \Leftrightarrow d), (0 \Leftrightarrow 0) \wedge (c \Leftrightarrow d) = b \to 0, (c \Leftrightarrow d)$
  Itt észrevehetjük, hogy $f_{\overline{a},\overline{b}} = f_{a,b}$ (izomorf részfák).
- $f_{a,b} = c \to f_{a,b,c}, f_{a,b,\overline{c}} = c \to (1 \Leftrightarrow d), (0 \Leftrightarrow d)$
- $f_{a,b,c} = d \to f_{a,b,c,d}, f_{a,b,c,\overline{d}} = d \to (1 \Leftrightarrow 1), (1 \Leftrightarrow 0) = d \to 1, 0$ (terminális csomópontok).
- $f_{a,b,\overline{c}} = d \to f_{a,b,\overline{c},d}, f_{a,b,\overline{c},\overline{d}} = d \to (0 \Leftrightarrow 1), (0 \Leftrightarrow 1) = d \to 0, 1$ (terminális csomópontok).

Az eredményül kapott ROBDD az ábrán látható.



18. ábra. ROBDD konstrukciója

6.5.5. A ROBDD előállítása

Egy $f(x_1, x_2, ..., x_n)$ Boole-függvény bináris döntési fa alakban való ábrázolása a Shannon-felbontás alapján végezhető el. A ROBDD felépítésekor az azonos változó-sorrendezés figyelembe vétele mellett az izomorf részfák illetve csomópontok összevonására és a szükségtelen csomópontok redukálására is figyelnünk kell.

A ROBDD reprezentációjához használjuk a következő adatstruktúrát:

- A ROBDD csomópontjait a $0, 1, 2, 3, ...$ indexekkel azonosítjuk, ahol a 0 és az 1 az ROBDD leveleit jelölik.
- Az $x_1, x_2, ..., x_n$ változókat sorrendben az $1, 2, 3, ..., n$ indexekkel azonosítjuk.
- A ROBDD-t egy $T : u \longmapsto (i, l, h)$ táblázatban tároljuk, ahol u a csomópont indexe, i a változó indexe, l a 0 behelyettesítési ágon elérhető csomópont indexe, h pedig az 1 behelyettesítési ágon elérhető csomópont indexe. A táblázaton értelmezzük a következő műveleteket:
  - init(T) T kezdeti állapotát állítja be, csak a 0 és 1 csomópontok (levelek) vannak a táblázatban.
  - add(T,i,l,h):u egy új csomópontot készít az adott paraméterekkel, és ennek u indexét adja vissza.

  - var(T,u):i adja vissza az u csomópont változójának i indexét, low(T,u):l és high(T,u):h pedig a két behelyettesítési ágon megtalálható csomópont l illetve h indexét.

- Az ROBDD csomópontjainak kereséséhez egy $H : (i,l,h) \longmapsto u$ táblázatot is nyilvántartunk a következő műveletekkel:

  - init(H) egy üres H táblázatot állít elő.
  - member(H,i,l,h):t ellenőrzi, hogy az (i,l,h) hármas szerepel-e H-ban, és erről egy t logikai igaz/hamis értéket ad vissza.
  - lookup(H,i,l,h):u kikeresi az (i,l,h) hármast a táblázatban és visszaadja a hozzá tartozó u csomópont indexet.
  - insert(H,i,l,h,u) beilleszt egy új sort a táblázatba.

Ha egyértelmű, a hívásokból elhagyjuk a T illetve a H paramétert. A továbbiakban feltételezzük, hogy a ROBDD építéséhez szükséges eljárások ezeken az adatstruktúrákon dolgoznak. Kezdjük a "csomópont építő" mk(i,l,h) függvénnyel, ami a következőképpen működik:

- Ha $l = h$, azaz azonos csomópontba vezetne a két él, akkor nem kell csomópontot létrehozni (redukálható a csomópont), bármelyik ág indexét vissza lehet adni.
- Ha a H táblázatban benne van már egy (i,l,h) hármassal jellemezhető csomópont, akkor sem kell újat létrehozni: van izomorf részfa, ennek az indexét lehet visszaadni.
- Ha nincs H-ban ilyen (i,l,h) hármas, akkor létre kell hozni az új csomópontot T-ben és a megfelelő sort H-ban, majd eztán visszaadni az így létrehozott csomópont indexét.

A pszeudo-kód:

```
mk(i,l,h){
      if l=h then return l;
      else if member(H,i,l,h) then return lookup(H,i,l,h);
      else {u=add(T,i,l,h); insert(H,i,l,h,u); return u;}
}
```

Ezután már a ROBDD-t építő build(f) függvényt készíthetjük el. Ez a Shannon-felbontást használja rekurzív módon, a redukció érdekében a fenti mk(i,l,h) függvényt hívogatva. A felbontást a változók indexének sorrendjében végezzük. A rekurzív hívások miatt egy build'(f,i) függvényt használunk, ami egy $t(x_i, x_{i+1}, ..., x_n)$ függvényt dolgoz fel:

```
build'(t,i){
      if i>n then if t=false then return 0 else return 1;
      else { v0=build'(t[0/x_i], i+1); v1=build'(t[1/x_i], i+1);
            return mk(i,v0,v1);}
}
```

Ezután a build(f) függvény már egyszerű:

```
build(f){
      init(T); init(H);
      return build'(f,1);
}
```

6.5.6. Műveletek ROBDD-ken

A Boole-függvényeken végzett műveleteket közvetlenül a ROBDD-ken hajthatjuk végre: a függvények ROBDD reprezentációi segítségével az eredmény ROBDD reprezentációját kapjuk meg. Feltétel, hogy a két függvény változói azonosak és a ROBDD-kben azonos sorrendben vannak felvéve.

A műveletek elvégzésének alapját a következő azonosságok képezik (itt op egy Boole-operátort jelöl):

$$\begin{aligned} f \,\mathrm{op}\, t &= (x \to f_x, f_{\overline{x}}) \,\mathrm{op}\, (x \to t_x, t_{\overline{x}}) = x \to (f_x \,\mathrm{op}\, t_x),\ (f_{\overline{x}} \,\mathrm{op}\, t_{\overline{x}}) \\ f \,\mathrm{op}\, t &= (x \to f_x, f_{\overline{x}}) \,\mathrm{op}\, t = x \to (f_x \,\mathrm{op}\, t),\ (f_{\overline{x}} \,\mathrm{op}\, t) \\ f \,\mathrm{op}\, t &= f \,\mathrm{op}\, (x \to t_x, t_{\overline{x}}) = x \to (f \,\mathrm{op}\, t_x),\ (f \,\mathrm{op}\, t_{\overline{x}}) \end{aligned}$$

Az azonosság alapján rekurzívan hozhatjuk létre az $f \;\mathrm{op}\; t$-hez tartozó ROBDD csomópontjait app(op,i,j) hívások segítségével, ahol i és j a művelet operandusainak ROBDD-jében a csomópontok.

Ez n változó esetén 2^n hívást igényelhetne. A gyorsítás érdekében egy G(i,j) táblázatot is használunk, ami a már kiszámolt app(op,i,j) eredményét tartalmazza (vagy üres, ha nem volt még ilyen). Így tipikusan $O(|f|\,|t|)$ műveletszám adódik.

Az algoritmus négy esetet különböztet meg:

- Mindkét csomópont levél: ekkor egy új levél hozható létre az operátort alkalmazva.
- Ha a csomópontokhoz tartozó változó indexe azonos, akkor a 0 és az 1 ágak (rekurzívan) párosíthatók az app() alkalmazásával; ez a fenti első azonosság alapján történik.
- Ha az egyik csomóponthoz tartozó változó indexe kisebb, akkor ezt párosítjuk a nagyobb változóindexű csomópont 0 és 1 ágaival a fenti $(x \to f_x, f_{\overline{x}}) \,\mathrm{op}\, t = x \to (f_x \,\mathrm{op}\, t),\ (f_{\overline{x}} \,\mathrm{op}\, t)$ azonosságot kihasználva.

A pszeudo-kód:

```
app(op,u1,u2){
    if G(u1,u2) ≠ empty then return G(u1,u2);
    else if u1 in {0,1} and u2 in {0,1} then u=op(u1,u2);
    else if var(u1)=var(u2) then u=mk(var(u1), app(op, low(u1),low(u2)),
        app(op, high(u1),high(u2)));
    else if var(u1) < var(u2) then u=mk(var(u1), app(op, low(u1),u2),
        app(op, high(u1),u2));
    else u=mk(var(u2), app(op, u1,low(u2)), app(op, u1,high(u2)));
    G(u1,u2)=u;
    return u;
}
```

Így a műveletet ténylegesen elvégző apply(op,f,t) függvény a következő:

```
apply(op,f,t){
    init(G);
    return app(op,f,t);
}
```

Részletesen ismertetjük még a restrict() függvényt, ami azt a ROBDD alakot állítja elő, ami egyes változók konstans értékekkel való behelyettesítésével adódik (ez szükséges például az egzisztenciális absztrakció számításához). Ennek alapesete, amikor egy x_j változó helyére kell $b \in \{0,1\}$ értéket helyettesíteni. Ha $b = 1$, akkor az u indexű x_j helyére high(u) kötendő be, ha $b = 0$, akkor pedig low(u). Az így adódó átkötés után felsőbb szintű csomópontok lehet, hogy redukálhatók lesznek, ezért az mk() hívásokat kell használnunk.

A pszeudo-kód:

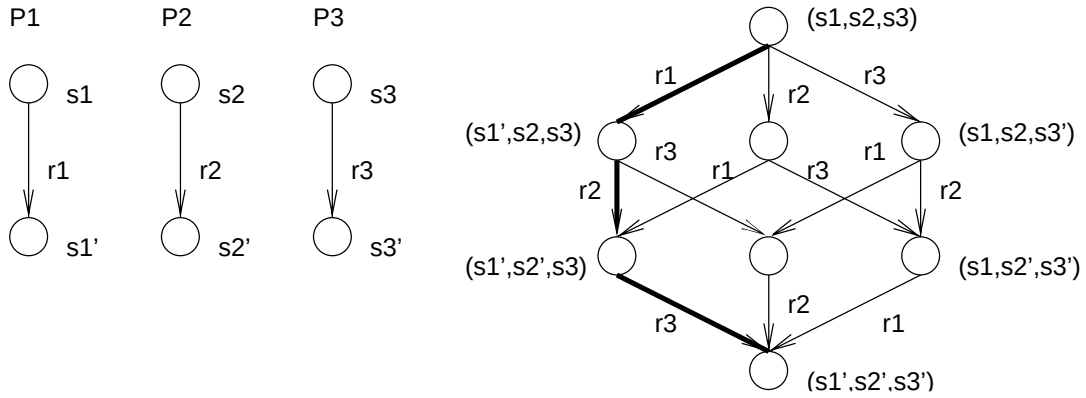
```
restrict(u,i,b){
    if var(u) > j then return u;
    else if var(u) < j then return mk(var(u),restrict(low(u),j,b),restrict(high(u),j,b));
    else if b=0 then return restrict(low(u),j,b) else return restrict(high(u),j,b);
}
```

6.5.7. Modell ellenőrzés ROBDD segítségével

Az előző alfejezetekben a modell ellenőrzést visszavezettük a karakterisztikus függvényeken végzett műveletekre. Ezeket ROBDD alakban kezelhetjük, a logikai műveleteket is közvetlenül a ROBDD reprezentáción elvégezve. Így a modell ellenőrzés tárigénye csökken (nagyobb állapotterű rendszerek ellenőrizhetők a kompakt reprezentáció miatt), és viszonylag gyors algoritmusok állnak rendelkezésre. Az iteráció befejezésének vizsgálatát megkönnyíti, hogy a ROBDD kanonikus alak.

6.6. Részleges rendezési technikák

A részleges rendezési technikák alapgondolata a konkurens rendszerek komponenseiben előforduló független állapotátmenetek átlapolt, sokféle sorrendben lehetséges, de ugyanahhoz az eredményhez vezető végrehajtásából eredő állapottér-méret növekedés elkerülése. E cél érdekében a sokféle végrehajtási útvonal helyett csak útvonalak egy *reprezentatív halmazát* veszik figyelembe a modell ellenőrzés során. Három független átmenet (r_1, r_2, r_3) átlapolt végrehajtásából adódó állapottérre mutat példát az 19. ábra jobb oldala. Ha az ellenőrizendő tulajdonság megengedi, hogy csak egy reprezentatív útvonalat válasszunk (pl. az ábrán a vastagon kihúzott útvonalat), akkor jóval kevesebb állapotot kell felvennünk. A kérdés tehát úgy merül fel, milyen tulajdonságok esetén és milyen reprezentatív útvonalakat vehetünk figyelembe?



19. ábra. Független állapotátmenetek átlapolt végrehajtása

Vezessük be a következő jelöléseket:

- $\mathrm{enabled}(s)$ jelöli egy s állapot *engedélyezett* átmeneteinek halmazát. Ez a konkurens komponensek (processzek) lokális állapotátmeneteiből tevődik össze; a lokális átmenet engedélyezettségét a lokális "programszámláló" és más helyi feltételek dönthetik el (kommunikáció, szinkronizáció stb.).

- $\mathrm{ample}(s) \subseteq \mathrm{enabled}(s)$ jelöli az s-ből induló reprezentatív átmenetek halmazát. A redukciót az jelenti, hogy az összes engedélyezett átmenet helyett csak ezeket vesszük figyelembe a globális állapottér építése során.

A teljes állapottér felvétele és az utólagos redukció (azaz $\mathrm{ample}(s)$ halmazok utólagos meghatározása) természetesen nem célszerű. Amikor a modell ellenőrzés során – tipikusan PLTL kifejezések ellenőrzésekor – felveszik a konkurens rendszer globális állapotterét, akkor általában mélységi keresést (DFS, Depth First Search) folytatnak: egy globális állapotban az engedélyezett átmeneteket keresik meg, ezek közül egyet kiválasztanak és ezen az úton haladnak tovább, az aktuális állapotot pedig verembe helyezik, és a választott út lezárása után térnek vissza a többi átmenet későbbi feldolgozására (új utak indítására). Amikor a DFS során egy s állapothoz érünk, akkor ott *lokálisan* kell a további útvonalak szempontjából számításba jövő $\mathrm{ample}(s)$ halmazt kiválasztani. A következők a követelmények:

- A modell ellenőrzés korrekt eredményt adjon (a modell ellenőrzés eredménye ne legyen érzékeny az elhagyott útvonalakra).

- A gráf mérete legyen kisebb (ez a redukció célja).

- Az $\mathrm{ample}(s)$ kiválasztásához szükséges számítási teljesítmény legyen kicsi (lehetőleg időt is nyerjünk).

6.6.1. Felhasznált tulajdonságok és célkitűzés

A követelmények teljesítéséhez a következő tulajdonságokat kell definiálnunk:

- *Független átmenetek*: A független átmenetek egy $I \subseteq R \times R$ relációt képeznek, amire a következők teljesülnek:

  - szimmetrikus, nem reflexív;
  - ha $(r_1, r_2) \in I$ akkor egyik sem tiltja le a másikat: ha $r_1, r_2 \in \mathrm{enabled}(s)$ akkor $r_1 \in \mathrm{enabled}(r_2(s))$, itt $r_2(s)$ jelöli az r_2 átmenet által s-ből elért állapotot (a szimmetria miatt $r_2 \in \mathrm{enabled}(r_1(s))$ is);
  - ha $(r_1, r_2) \in I$ akkor különböző végrehajtási sorrendjük ugyanahhoz az állapothoz vezet: $r_1(r_2(s)) = r_2(r_1(s))$.

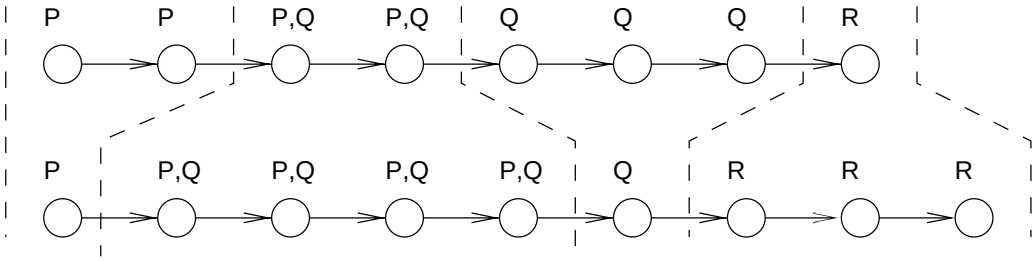
  Felvetődik, hogy a független állapotátmenetek közül elég lenne csak egyet kiválasztani az $\mathrm{ample}(s)$ halmazba. Ez viszont a következő problémákhoz vezethet (23. ábra):

  1. példa: Az ellenőrizendő PLTL kifejezés függhet az r_1 vagy r_2 választástól, azaz $r_1(s)$ illetve $r_2(s)$ állapotoktól (pl. más címkézése van ezeknek az állapotoknak).
  2. példa: Lehetséges, hogy $r_1(s)$ illetve $r_2(s)$ állapotokból más-más további átmenetek is indulnak, így a bejárt utak nem lesznek egyenértékűek.

  Ki kell tehát még terjeszteni a figyelembe veendő tulajdonságok körét.

- *Láthatatlan átmenet*: Egy $r_1 = (s, s')$ átmenet láthatatlan, ha $L(s) = L(s')$. Ez azt jelenti, hogy a címkézés nem módosul az elért állapotban a kiindulási állapothoz képest (erre nem lesz érzékeny az ellenőrizendő kifejezés, az A_p automata nem "látja" az átmenetet). Itt csak azokat az $AP' \subseteq AP$-beli címkéket kell tekintenünk, amik érdekesek az ellenőrizendő tulajdonság szempontjából.

- *Dadogás*: Egy útvonalban lehetségesek – a láthatatlan átmenetekből adódóan – egymás után következő, azonosan címkézett állapotok. Ezekből blokkokat állíthatunk össze, ezeket *dadogó blokkoknak* nevezhetjük. Két útvonal dadogó ekvivalens, jelölése $\pi_1 \sim_{st} \pi_2$, ha azonos dadogó blokkok, azonos sorrendben fordulnak elő a két útvonalon. Ez az ekvivalencia tehát eltekint attól, hogy egy-egy blokkban hány állapot található. Egy példát mutat a 20. ábra.



20. ábra. Dadogó ekvivalens útvonalak

  Két Kripke-struktúra dadogó ekvivalens, jelölése $M_1 \sim_{st} M_2$, ha a kezdőállapotok azonosak, és minden M_1-beli π_1 útvonalhoz található olyan M_2-beli π_2 útvonal, hogy $\pi_1 \sim_{st} \pi_2$.

  Egy p PLTL kifejezés *dadogó invariáns*, ha minden $\pi_1 \sim_{st} \pi_2$ útvonalra igaz, hogy $\pi_1 \models p$ a.cs.a. $\pi_2 \models p$.

A dadogó ekvivalens PLTL kifejezésekre vonatkozóan két tételt mondhatunk ki:

- A $PLTL_{-X}$ kifejezések dadogó invariánsak. Itt $PLTL_{-X}$ jelöli azt a lineáris idejű temporális logikát, amit a már megismert PLTL logikából kapunk az X operátor elhagyásával. Nyilvánvalóan az X operátor "érzékeny" az egymás utáni azonosan címkézett állapotok számára is, amitől a dadogó blokkokban elvonatkoztattunk.

- Ha egy PLTL kifejezés dadogó invariáns, akkor az egy $PLTL_{-X}$ kifejezés.

Azt tűzzük ki célul, hogy a részleges rendezés redukció olyan redukált Kripke-struktúrát eredményezzen, amely az eredetivel dadogó ekvivalens. Ezek után a $PLTL_{-X}$ kifejezéseket ellenőrizhetjük a redukált struktúrán.

6.6.2. Feltételek a reprezentatív átmenet-halmaz konstruálásához

Egy adott s állapotban az $\mathrm{ample}(s)$ halmaz konstruálása négy feltételnek kell eleget tegyen ahhoz, hogy a redukált Kripke-struktúra az eredetivel dadogó ekvivalens legyen. Ezeket a feltételeket adjuk meg az alábbiakban, kitérve a feltételek mögötti meggondolásokra. A szükségesség és elégségesség formális bizonyítását viszont nem tárgyaljuk.

- (C0) $\mathrm{ample}(s) = \emptyset$ a.cs.a. $\mathrm{enabled}(s) = \emptyset$. Ez egy magától értetődő kritérium.

- (C1) Az eredeti Kripke-struktúrában, minden útvonal mentén s-ből indulva, igaz: Egy $r \in \mathrm{ample}(s)$-től *függő* átmenet nem hajtható végre anélkül, hogy előtte egy $\mathrm{ample}(s)$-beli átmenetet végre ne hajtanánk.

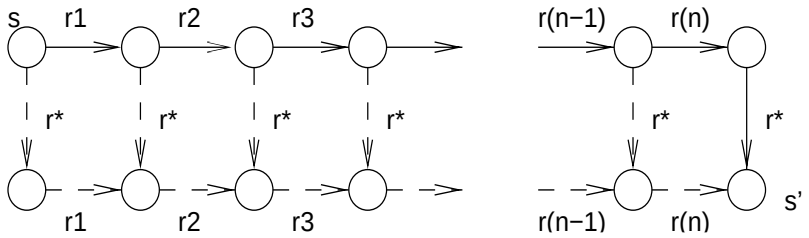
  Ennek a feltételnek rögtön van egy fontos következménye: Az $\mathrm{enabled}(s) \setminus \mathrm{ample}(s)$ halmazbeli átmenetek függetlenek az $\mathrm{ample}(s)$-beli átmenetektől (ellenkező esetben sérülne C1).

  Ha az $\mathrm{ample}(s)$ választása C0 és C1 figyelembevételével történik, akkor a lehetséges kimaradó (le nem fedett) útvonalak a következő formájú útvonalakra szűkülnek:

  – $r_1, r_2, ..., r_n, r^*$ ahol $r_1, r_2, ..., r_n$ függetlenek az $\mathrm{ample}(s)$-beli átmenetektől és $r^* \in \mathrm{ample}(s)$.
  – $r_1, r_2, ..., r_i, ...$ végtelen útvonal, ahol $r_1, r_2, ..., r_i, ...$ függetlenek $\mathrm{ample}(s)$-beli átmenetektől.

  Az első esetben a függetlenség miatt beláthatjuk, hogy s-ből az $r_1, r_2, ..., r_n, r^*$ útvonal ugyanabba az s' állapotba kell vezessen, mint az $r^*, r_1, r_2, ..., r_n$ útvonal: a függetlenség definíciójában szereplő harmadik pontot kell alkalmaznunk sorban a $r_n, r_{n-1}, ..., r_1$ átmenetekre, így r^*-et "előrehozhatjuk", ld. 21. ábra. Ez utóbbi útvonal pedig már $\mathrm{ample}(s)$-beli átmenettel indul.

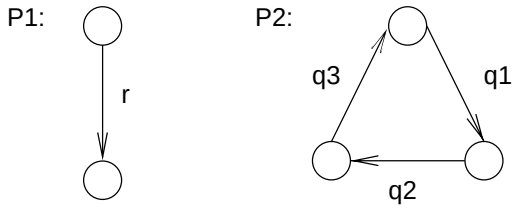
  Ezek után a $r_1, r_2, ..., r_n, r^*$ útvonal is lefedett lesz a modell ellenőrzés által, ha $r_1, r_2, ..., r_n, r^* \sim_{st} r^*, r_1, r_2, ..., r_n$. Ezt biztosítja a következő C2 szabály.



21. ábra. Azonos állapotba vezető útvonalak

- (C2) Ha $\mathrm{ample}(s) \neq \mathrm{enabled}(s)$, akkor minden $r \in \mathrm{ample}(s)$ átmenet láthatatlan kell legyen.

  Ez a szabály azt is biztosítja, hogy $r_1, r_2, ..., r_i, ... \sim_{st} r^*, r_1, r_2, ..., r_i, ...$ azaz a fent említett második típusú útvonal is lefedett lesz a modell ellenőrzés által.

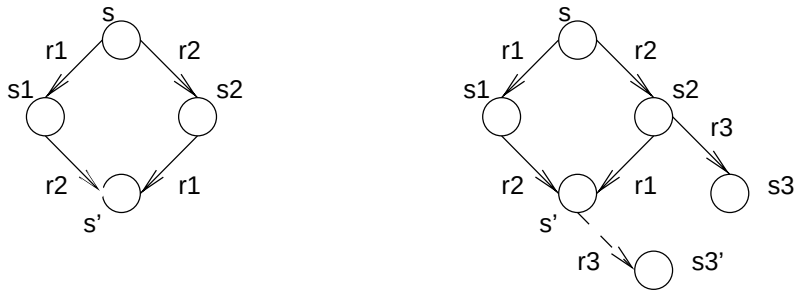


22. ábra. Ciklus kialakulása

Még egy lehetőséget hagytunk ki: mi történik, ha ciklus alakul ki $\mathrm{ample}(s)$-beli átmenetekből, és a ciklus miatt maradnak ki átmenetek? Nézzünk erre egy példát! A 22. ábrán látható két processz; r egy r_1, r_2, r_3-tól független, nem láthatatlan átmenet, míg r_1, r_2, és r_3 láthatatlanok, egymástól függőek. Előfordulhat, hogy az $\mathrm{ample}(s)$ halmazba egymás után a r_1, majd r_2, és r_3 átmeneteket választjuk, és r-et "későbbre hagyjuk". Csakhogy a ciklus záródik, így lehet, hogy r kimarad. Ennek elkerülését fogalmazza meg a C3 feltétel:

- (C3) Olyan ciklus nem megengedett, amelyben található olyan állapot, ahol egy r átmenet engedélyezett, de ez nincs benne a ciklus állapotaihoz tartozó egy $\mathrm{ample}()$ halmazban sem.

Ezek után illusztrációként nézzük meg, hogy fentebb, a függetlenség definíciója után említett két "ellenpélda" esetén hogyan biztosítják a szabályok a korrektséget. A 23. ábra fog segíteni a jelölésekben.



23. ábra. Példák a szabályok alkalmazására

1. példa: Tegyük fel, hogy a két független átmenet közül r_1 van benne $\mathrm{ample}(s)$-ben. A C2 szabály miatt r_1 láthatatlan, ezért biztos, hogy $r_1, r_2 \sim_{st} r_2, r_1$. (Ha s címkéje P és s_2 címkéje Q, akkor r_1 láthatatlansága miatt s_1 címkéje is P lesz, s' címkéje pedig Q, tehát előállnak a dadogó blokkok.)

2. példa: Ez esetben is tegyük fel, hogy $r_1 \in \mathrm{ample}(s)$. C1 miatt r_1 és r_3 függetlenek kell legyenek. Így, mivel r_2 nem tiltja le r_3-at, r_3 engedélyezett az s' állapotban is. Mivel pedig C2 miatt r_1 láthatatlan, ezért fennáll, hogy $r_2, r_3 \sim_{st} r_1, r_2, r_3$. (Ha s címkéje P, s_2 címkéje Q, s_3 címkéje pedig R, akkor r_1 láthatatlansága miatt s_1 címkéje is P, s' címkéje pedig – r_2 miatt – Q kell legyen. Ebből adódóan – az s_2-ből induló r_3 mintájára – s'_3 címkéje is R kell legyen, a dadogó blokkok tehát előálltak.)

6.6.3. Gyakorlati megvalósítás

A szabályok megadása még nem jelenti azt, hogy hatékony algoritmusunk is van az $\mathrm{ample}()$ halmaz konstruálásához. Az egyes szabályok ellenőrzése ugyanis eltérő számítási igényt jelent. Ha a szabály ellenőrzésének számítási igénye nagy, akkor kompromisszumot kell kötni: Olyan $\mathrm{ample}()$ halmazt kell konstruálni, ami kis számítási igénnyel megtehető, garantáltan teljesíti a szabályt, de nem biztos, hogy optimális. Így a redukált Kripke-struktúra sem feltétlenül lesz az elméletileg lehetséges legkisebb méretű, de sok esetben így is jelentős megtakarításokat eredményez a modell ellenőrzés szempontjából.

- C0 ellenőrzése lineáris számítási igényű, könnyen ellenőrizhető.

- C2 szintén egyszerűen ellenőrizhető, az engedélyezett átmeneteket kell végignézni, és ebből adódik a potenciális $\mathrm{ample}()$ halmaz.

- C1 ellenőrzése olyan komplex, mint az elérhetőségi probléma (tehát annak eldöntése, hogy egy globális állapot elérhető-e egy adott kezdőállapotból). Ez megengedhetetlenül nagy számítási igényt jelentene. Ezért kompromisszumot kell kötni egy garantált, de adott esetben nem optimális $\mathrm{ample}()$ konstruálással. Ez rendszerint úgy történik, hogy egy konkurens P_i processz engedélyezett T_i átmeneteit veszik, és ellenőrzik a következőket: Nincs más P_j processzben olyan átmenet, ami engedélyezett és függő T_i átmeneteitől, vagy aminek végrehajtása engedélyezetté tesz P_i-ben egy T_i-ben most nem szereplő függő átmenetet. Ha nincs ilyen átmenet, akkor $\mathrm{ample}() = T_i$,

egyébként a következő processzre kell áttérni (legrosszabb esetben pedig minden processzt teljesen kibontani).

- C3 ellenőrzése úgy történik, hogy egy szigorúbb C3' feltétellel helyettesítik az eredeti, nagy számításigényű ellenőrzést: Ha $\mathrm{ample}(s) \neq \mathrm{enabled}(s)$, akkor nem szabad, hogy legyen olyan él, ami már felvett (tehát a DFS veremben lévő) állapotba mutat. Ez a feltétel a verem vizsgálatával ellenőrizhető. Ha van ilyen él (azaz ciklus), akkor teljesen ki kell bontani az adott állapotot: $\mathrm{ample}(s) = \mathrm{enabled}(s)$. Ez biztosítja azt, hogy minden ciklusban van olyan állapot, ahol $\mathrm{ample}(s) = \mathrm{enabled}(s)$.

6.6.4. A részleges rendezés redukció hatékonysága

Ha egy konkurens rendszerben sok független állapotátmenet van, és ezek teljesítik a redukciós szabályokat, akkor a részleges rendezés eredménye garantált (nem függ "heurisztikus" tényezőktől, mint pl. az ROBDD esetén a változók sorrendezése). A tapasztalatok szerint tipikusan egy nagyságrendnyi redukció érhető el az állapottérben. Sok, viszonylag független résztvevőt tartalmazó protokollok esetén a résztvevők függvényében exponenciális méretű állapottér rendszerint polinom méretű lesz a redukció után.

Egyirányú gyűrű kommunikációs struktúrával összekapcsolt processzek esetén egy, a vezetőválasztáshoz alkalmazott protokoll modell ellenőrzése során a következő adatok voltak mérhetők (a részleges rendezést alkalmazó SPIN modell ellenőrzőben):

Processzek száma	Redukció nélkül		Redukcióval	
	Állapotok	Időigény	Állapotok	Időigény
3	15 929	13,8 s	1 435	0,6 s
4	522 255	9,3 perc	8 475	3,5 s
5		>40 óra	57 555	28,7 s
6			434 083	4,1 perc

6.7. A modell ellenőrzés komplexitása

A tárgyalt temporális logikák különböznek a modell ellenőrzés idő- illetve tárkomplexitásában. Bizonyítás nélkül megadunk néhány tételt, amelyek erre vonatkozóan adnak támpontot.

Egy p PLTL kifejezés modell ellenőrzéséről az $M = (S, R, L)$ struktúrán a következők mondhatók:

- A modell ellenőrzés időkomplexitása legrosszabb esetben négyzetes a struktúra méretével és exponenciális a PLTL kifejezés méretével: $O(|S|^2 \times 2^{|p|})$.
  - A struktúra méretével való négyzetes komplexitás abból adódik, hogy az $|S|$ állapotú rendszerben legrosszabb esetben $|S|^2$ állapotátmenet lehet. Ha az átlagos esetet vesszük figyelembe, akkor a struktúra méretével csak lineáris időkomplexitásról beszélhetünk.
  - Az exponenciális komplexitás abból adódik, hogy a p PLTL kifejezésből az A_p automata előállítása során az automata állapotait a rész-kifejezésekkel címkézzük (ez a tabló előállítása), a rész-kifejezések ("részhalmazok") száma pedig $O(2^{|p|})$. Az exponenciális komplexitás riasztóan hat, de szerencsére általában a struktúra mérete nagy, a PLTL kifejezések pedig viszonylag rövidek, így a komplexitás kezelhető marad.
  - A szinkron szorzat automata állapotterének mérete tehát $O(|S|^2 \times 2^{|p|})$, és mivel a nyelvi üresség ellenőrzésének időigénye lineáris a szorzat automata állapotterének méretével, ezért a modell ellenőrzés komplexitása is ilyen.

- A PLTL modell ellenőrzés PSPACE-teljes, tehát az szükséges tár mérete polinomiális a struktúra és a PLTL kifejezés méretével.

Egy p BTL kifejezés modell ellenőrzéséről a következők mondhatók:

- CTL kifejezések esetén a modell ellenőrzés időkomplexitása legrosszabb esetben négyzetes a struktúra méretével és lineáris a CTL kifejezés méretével: $O(|S|^2 \times |p|)$.

  A lineáris függés abból adódik, hogy itt a p kifejezést a szintakszis szabályai alapján bontjuk részkifejezésekre, és külön-külön kiszámítjuk az ezeket kielégítő állapothalmazokat (iterációval). Az iteráció komplexitása nem szorzódik, mivel a CTL kifejezések fixpont karakterisztikáiban nem szerepelnek egymásba ágyazott, egymásra ható váltakozó legkisebb illetve legnagyobb fixpont számítások.

  Ez a PLTL-énél kedvezőbb komplexitást jelent, ami némiképp meglepő, mert a CTL bonyolultabbnak tűnik a PLTL-nél. Megjegyzendő persze, hogy egy CTL-ben adott tulajdonság, persze csak ha kifejezhető PLTL-ben, általában rövidebb PLTL formulát eredményez (hiszen az útvonal kvantorokat kell elhagyni), így a CTL előnye eltűnik. Azt be is bizonyították, hogy egy követelményre, ami PLTL és CTL segítségével egyaránt megfogalmazható, igaz az, hogy a PLTL kifejezés soha nem hosszabb, mint a CTL kifejezés, és sok esetben exponenciálisan rövidebb.

- FairCTL esetén a modell ellenőrzés időkomplexitása legrosszabb esetben $O(|S|^2 \times |p| \times |\phi|)$, ahol ϕ a méltányosság kifejezésére szolgáló útvonal-kifejezés.

- Ha egy PLTL-hez van egy modell ellenőrző algoritmusunk, akkor ahhoz a BTL-hez, ami a PLTL modális operátorait tartalmazza, szintén van egy azonos komplexitású algoritmusunk. Ez ésszerűvé teszi a CTL* használatát.

- CTL* esetén a modell ellenőrzés PSPACE teljes. (Hiszen a CTL* esetén az A és E operátorokat követheti egy PLTL kifejezés, amiről láttuk, hogy PSPACE teljes.)

6.8. Temporális logikai modell ellenőrzés előnyei és korlátjai

A temporális logikai modell ellenőrzés előnyei a következőkben fogalmazhatók meg:

- Jól kidolgozottak a matematikai alapok. A követelmények megfogalmazására illetve az állapottér kezelésére hatékony technikák állnak rendelkezésre. Csodát természetesen nem várhatunk: bár 10^{120} nagyságú állapotterek esetén végeztek sikeres modell ellenőrzést, ezek a nagyságrendek csak reguláris struktúrájú, tehát ROBDD segítségével ”jól tömöríthető” állapotterek esetén kezelhetők.

- Általánosan alkalmazható véges állapotterű rendszerekben, így hardver, szoftver, protokollok esetén is.

- Részleges ellenőrzést támogat, mivel egyenként ellenőrizhetők az egyes temporális logikai követelmények. Nem szükséges egy teljes, hibátlan ”referencia terv” az ellenőrzés végrehajtásához.

- A modell ellenőrzési feladat automatikus megoldására kereskedelmi illetve akadémiai eszközök léteznek. A tervező rutinszerűen, ”egy gombnyomásra” alkalmazhatja a modell ellenőrző eszközöket (ezek nem igényelnek interakciót, intuitív beavatkozást).

A hátrányok között a következők említhetők:

- Elsősorban vezérlés-orientált alkalmazásokhoz való. Komplex adatstruktúrák és az adatkezelés modellezése rendszerint kezelhetetlenül nagy állapotterekhez vezet.
- A gyakorlatban az állapottér robbanás erősen korlátozza a modell ellenőrzéssel vizsgálható rendszerek méretét. (Elosztott rendszerekben a rendszer globális állapotterének mérete legrosszabb esetben az egyes komponensek állapotterei méretének a szorzata.) Az általában nem garantálható, hogy akár részleges rendezéssel, akár ROBDD-k használatával sikerül elég kompakt formában reprezentálni az állapotteret.
- Az állapottér méretének kordában tartása érdekében szükséges a modell és a követelmények megfelelő absztrakciós szintjének megtalálása, ami jelenleg még intuitív és sok tapasztalatot igénylő feladat.
- A modell ellenőrzés eredménye alapján nehéz általánosítani: ha pl. egy algoritmus 1 és 2 processz esetén működik, nem biztos, hogy $3, 4, 5, ..., n$ processzor esetén is működni fog. Ugyanakkor a modell ellenőrzés eredménye hozzájárulhat az általános tételek kimondásához és bizonyításához, pl. egy induktív bizonyítás esetén a kiindulási esetben érvényes tulajdonság belátásával.
- A követelmények teljessége nem ellenőrizhető. Az egyes követelmények különálló megfogalmazásának lehetősége a részleges ellenőrzés szempontjából előnyt jelentett, de a teljesség szempontjából hátrányként is jelentkezik.

Külön hangsúlyoznunk kell, hogy a modell ellenőrzés – mint verifikációs technika – esetén a *modell* (és nem a valóságban implementált rendszer) ellenőrzése történik meg, tehát a tervező szempontjából a modell ellenőrzés eredménye csak annyira használható, amennyire a modell a valósághoz hű. A modell ellenőrzés a validációt nem helyettesíti.

A nemzetközi szakirodalomban nagyszámú leírást találunk ipari környezetben is sikeres modell ellenőrzésről. Ezek közül többet a bevezető fejezet ismertet. A példákból is kitűnik, hogy a modell ellenőrzés ipari alkalmazása terjedőben van a kritikus, hibák esetén nagy kockázatú rendszerek verifikációjának céljára. Az elterjedés nem utolsósorban az automatikus, ”egy gombnyomásra használható” modell ellenőrző eszközöknek köszönhető. Ezekből soroljuk fel a jelentősebbeket az alábbiakban:

- Az első két modell ellenőrző, az EMC és a Caesar, 1981-ben jelent meg.
- Az SMV volt az első olyan modell ellenőrző, ami ROBDD-ket használt az állapottér kezelésére. Ennek újabb, bővített verziója a nuSMV. Mindkét eszköz CTL modell ellenőrzést végez.
- Az 1991-től fejlesztett SPIN modell ellenőrző a részleges rendezés technikáját alkalmazza. Elsősorban protokollok és elosztott algoritmusok PLTL alapú ellenőrzésére bizonyult hatékonynak.
- A Concurrency Workbench egy olyan logikát, az úgynevezett μ-kalkulust támogat, aminek segítségével CTL* és PLTL kifejezések egyaránt felírhatók. A μ-kalkulus közvetlenül tartalmazza a szemantikai alapú modell ellenőrzésnél megismert fixpont műveleteket.
- A HyTech rendszer azért terjedt el széles körben, mert hibrid rendszerek (amelyek diszkrét logikát és folytonos komponenseket is tartalmaznak) verifikációját támogatja.
- Az idő adatokat is tartalmazó modellek ellenőrzésére alkalmas eszközök a Kronos és az Uppaal.
- Közvetlenül a Verilog hardver leíró nyelvet támogatja a VIS modell ellenőrző, ami CTL kifejezéseket tud BDD-k felhasználásával ellenőrizni.

- Az iLogix cég nemrégen jelentette be, hogy Statemate nevű, állapottérképeket alkalmazó szoftver tervező és kódgeneráló eszközéhez automatikus modell ellenőrzést biztosít.
- A Java nyelvhez olyan eszközöket dolgoztak ki (pl. PathFinder, Bandera), amelyek a forráskód alapján képesek a vezérlési struktúra formális modelljének generálására és ennek modell ellenőrzésére.

A modell ellenőrzés esetén is bebizonyosodott az, ami a formális módszerek alkalmazásával kapcsolatban általában is állítható: a formális modellek elkészítése illetve ezek verifikálása a tervezési fázisban növeli a költségeket (többlet időt, szakértelmet igényel), de ez a költségnövekedés összességében általában kevesebb, mint ami a tesztelés, debuggolás és javítás során a kevesebb hiba miatt megtakarítható.