Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
Hibatűrő Rendszerek Kutatócsoport

Bevezető az UPPAAL használatába

Készített:
Darvas Dániel,
Horányi Gergő

Utolsó módosítás:
Vörös András
2017. április 21.

1 Bevezető

Ez a leírás rövid bevezetőül kíván szolgálni a Rendszertervezés szakkör résztvevői számára az UPPAAL rendszerről. A dokumentum épít a Behrmann, David és Larsen által írt UPPAAL segédletre [1]. A leírtak nem tudományos igényességűek és helyenként jelentősen leegyszerűsítettek, jelen leírás célja a gyors bevezetés példákon keresztül az UPPAAL világába.

1.1 Az UPPAAL-ról

Az UPPAAL [1] egy valós idejű rendszerek modellezésére és verifikációjára szolgáló rendszer, amelyet az Uppsala University és az Aalborg University fejleszt. Első változata 1995-ben jelent meg, jelenleg a 4.0-s verziónál tartunk.

2 Modellezés UPPAAL-ban

Az UPPAAL időzített automaták hálózatát képes modellezni. Az időzített automata egy véges állapotgép óraváltozókkal kiegészítve. A fejezet első részében a kiegészítések nélküli állapotgép-formalizmust tekintjük át, majd a későbbiekben bemutatjuk röviden a kibővítéseket is.

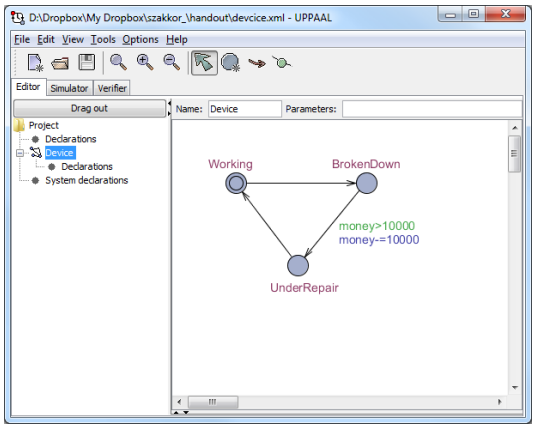
2.1 Állapotgépek UPPAAL-ban

Az UPPAAL által modellezett időzített automaták hálózata egymással konkurens működésű folyamatokból áll. Minden folyamat egy-egy véges állapotgéppel írható le (ha eltekintünk a kibővítésektől). Az egyes folyamatok állapotgépeinek leírása hasonló a már ismert UML-ben modellezhető State Machine-ekhez.

Az állapotgép alapelemei a következők:

- állapotok (helyek, locations): névvel rendelkeznek, közülük mindig pontosan egy aktív
- állapotátmenetek (transitions): a lehetséges állapotváltásokat jelölik ki, ezekhez feltételek és akciók társulhatnak

Egy állapotgépet mutat az UPPAAL főablakában az 1. ábra.



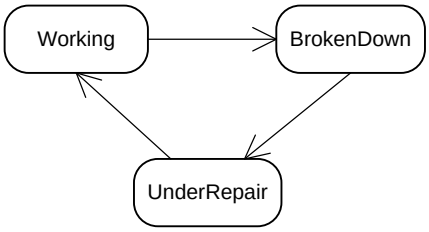
1. ábra Állapotgép UPPAAL-ban

A 2. ábra és a 3. ábra egy egyszerű állapotgépet mutat az UML és az UPPAAL formalizmusában. Az állapotgép egy rendszert ír le, amely a következő állapotokban lehet: *működő*, *hibás*, *javítás alatt álló*. Az állapotok közt lehetségesek átmenetek: egy működő rendszer elromolhat, majd egy hibás rendszernek elkezdődhet a javítása, melynek végén a rendszer újra működőképes lesz. Ezt

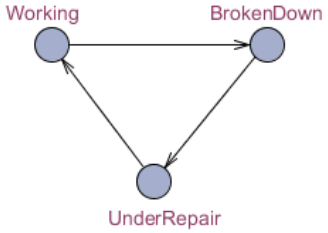
[1] Az UPPAAL akadémiai célokra ingyenesen elérhető a következő címen: http://www.uppaal.org/.

mutatják az ábrák nyilai is. Azonban pl. egy működő rendszer javítását nem engedjük meg („ami nem romlott el, azt nem kell megjavítani"), ezért ilyen irányú nyíl nem található az ábrákon.

Megjegyzés: egy állapotból több engedélyezett állapotátmenet is vezethet ki. Ilyen esetben véletlenszerűen választjuk ki a következő állapotot.

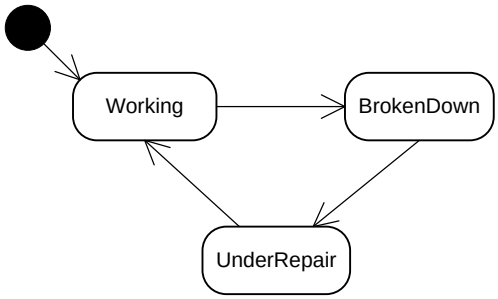


2. ábra Egyszerű UML State Chart példa

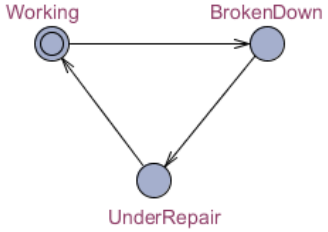


3. ábra Egyszerű UPPAAL állapotgép

Minden állapotgépre meg kell adnunk egy kezdőállapotot is. Ezt UPPAAL-ban dupla karikával jelöljük. Egy állapotot kezdőállapotnak dupla kattintás után az *Initial* mező kiválasztásával lehet beállítani.

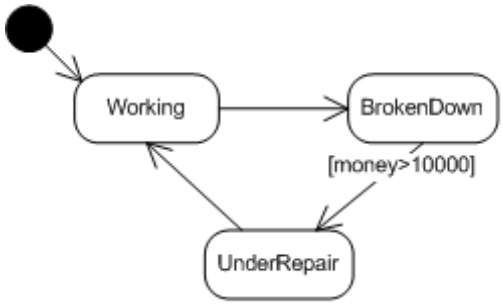


4. ábra Egyszerű UML State Chart példa kezdőállapottal

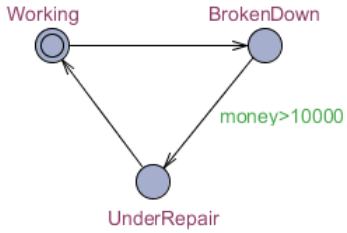


5. ábra Egyszerű UPPAAL állapotgép kezdőállapottal

Lehetőség van az állapotátmeneteket feltételekkel is ellátni. Például egy hibás eszköz javítása csak akkor kezdődhet meg, ha legalább 10000 pénz rendelkezésre áll. Erre UPPAAL-ban az *őrfeltételek* (*guard*) szolgálnak. Ezekben változókra tehetünk feltételeket. A példában specifikált feltételt „pénz>10000" formában tudjuk kifejezni és az állapotátmenetet jelképező élre duplán kattintva tudjuk megadni.



6. ábra Egyszerű UML State Chart példa őrfeltétellel



7. ábra Egyszerű UPPAAL állapotgép őrfeltétellel

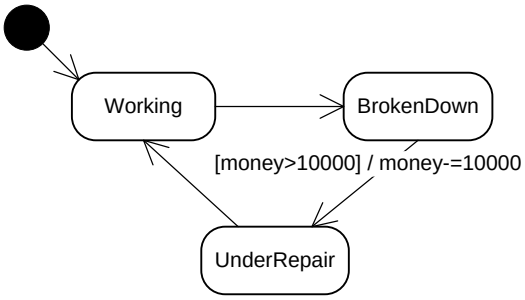
UPPAAL-ban az őrfeltételekben használt változókat deklarálni kell az automatához tartozó Declarations részen vagy a globális Declarations részen. A változók deklarálására egy C-szerű szintaxis használható, például jelen esetben:

```
int money = 25000;
```

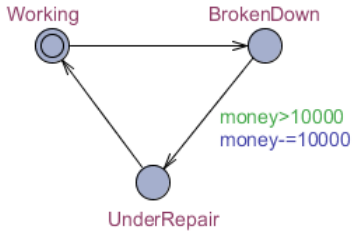
A változók deklarációjának szintaktikáját bővebben itt nem tárgyaljuk. Precíz definíciója megtalálható a [4] dokumentumban.

Az egyes állapotátmenetekhez nem csak feltételek, hanem az átmenet során végrehajtandó akciók is tartozhatnak. Például a javítás megkezdésekor fizessük ki a szerelőt, azaz csökkentsük a rendelkezésre álló pénzmennyiséget 10000-rel. Ezt az élre duplán kattintva az Update mezőben lehet megadni. Például jelen esetben így:

```
money -= 10000
```



8. ábra Egyszerű UML State Chart példa akcióval



9. ábra Egyszerű UPPAAL állapotgép akcióval

2.2 Szimuláció UPPAAL-ban

Az UPPAAL nem csak modellek szerkesztésére, hanem azok szimulációjára is lehetőséget nyújt. Ehhez előbb viszont meg kell értenünk egy jelentős különbséget az UML State Chart és az UPPAAL modell közt. Az UML-ben állapotgépeket példányok szintjén modellezhetünk, míg UPPAAL-ban típusok (sablonok, templatek) szintjén. Valójában mi UPPAAL-ban egy állapotgép-típust definiáltunk, amelyből jelenleg egyetlen példány sem létezik, így nincs mit szimulálnunk.

A példányosításhoz először nevezzük el az elkészített sablonunkat az ablak felső részén:

10. ábra Sablon elnevezése

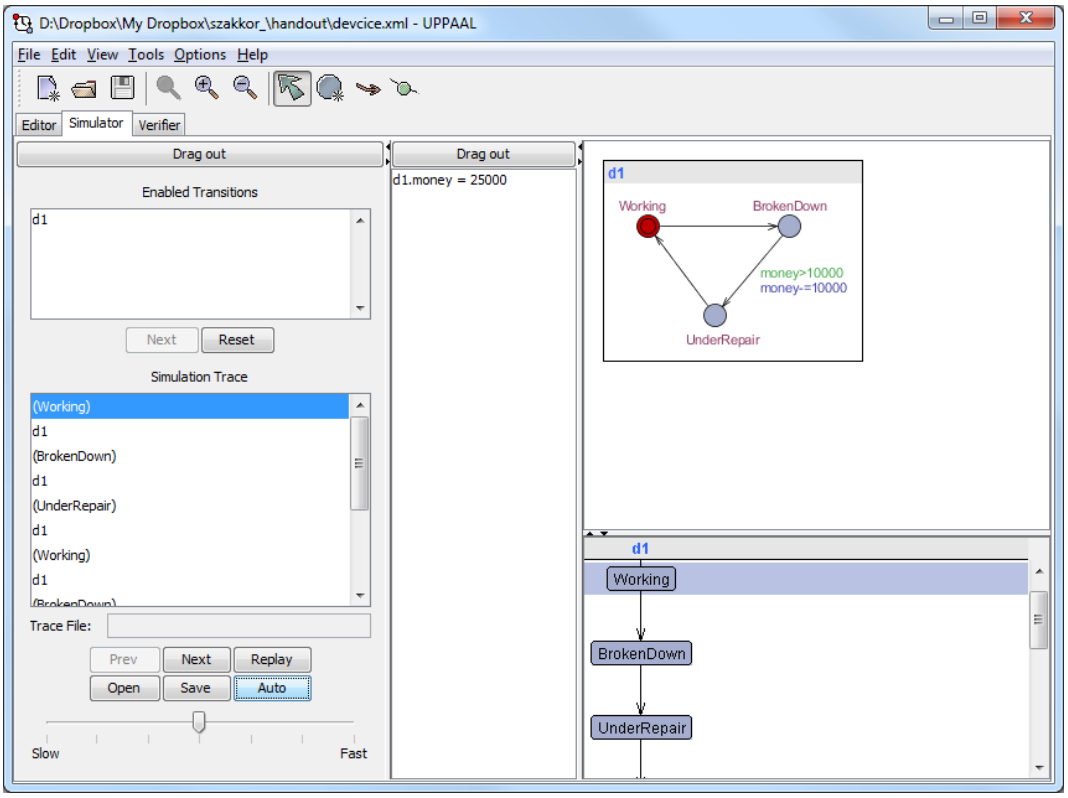
Ez után kattintsunk a bal oldali fastruktúrában a *System declarations* elemre! Itt példányosíthatjuk az egyes sablonainkat. Ahhoz, hogy készítsünk egy d1 példányt a Device sablonból, majd ezt egy rendszerbe foglaljuk, a következő kódot kell beírnunk:

```
// Place template instantiations here.
d1 = Device();

// List one or more processes to be composed into a system.
system d1;
```

Ez után már kipróbálhatjuk a rendszer szimulációját. Ehhez kattintsunk a *Simulator* fülre. A modell minden módosítása után megkérdezi az UPPAAL, hogy akarjuk-e a szimulátorban szereplő modellt frissíteni („Do you want to upload the model now?”), erre válaszoljunk *Yes*-szel.

Most már látjuk a d1 példányunkat. A pirossal jelölt állapot az állapotgép aktuális állapota. Tőle balra láthatjuk a deklarált változók aktuális értékét is. A Next gombra kattintva változtathatjuk az aktuális állapotot. Az állapotokat automatikusan is léptethetjük a képernyő alján lévő Auto gombra kattintva. Így láthatjuk, hogy idővel a futás megáll, hiszen nem lesz elegendő pénz a javítás megkezdéséhez.



11. ábra Az UPPAAL szimulátor felülete

2.3 *Időzítések*

Az UPPAAL az eddig ismertetetteken túl lehetőséget nyújt időzítések kezelésére is. Ehhez ún. óraváltozókat deklarálhatunk. Az óraváltozó egy logikai órát reprezentál. Pontos értékét tipikusan nem ismerjük, csak azt, hogy értéke milyen tartományok közt van.

Az óraváltozók fontos tulajdonságai:

- Az óraváltozók értéke monoton nő, hacsak nem állítjuk explicit vissza őket.
- Egy állapotban tetszőlegesen sok (de csak véges), vagy akár éppen 0 idő tölthető el. (Ennek korlátozására az invariánsok és speciális állapotok léteznek, de ezek ismertetésére nem térünk ki.)
    - Következménye: előfordulhat, hogy egy állapotból eleinte csak egy engedélyezett átmenet van, de az idő múlásával újak is engedélyezetté válnak és ezek egyike fog ténylegesen végrehajtódni.

Az óraváltozókat először deklarálnunk kell. Ahhoz, hogy létrehozzunk egy `c1` nevű óraváltozót, a következő kódot kell beírnunk:

```
clock c1;
```

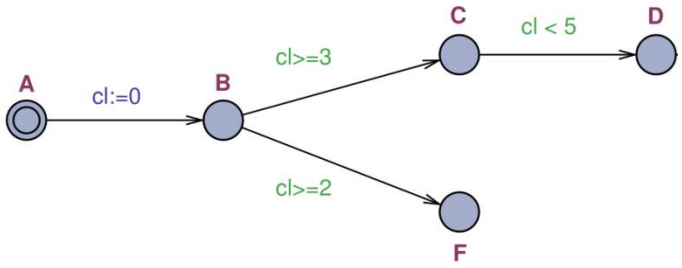
Ez után az óraváltozó értékére feltételeket tehetünk az őrfeltételekben, illetve az akciók során konkrét értéket állíthatunk be neki.

Az óraváltozók működése legkönnyebben egy példán keresztül (12. ábra) érthető meg. Amikor az A→B állapotátmenet lezajlik, a `cl` óra értéke 0 lesz. Így azt tudhatjuk, hogy amikor a rendszer B állapotban van, az óra értéke: $cl \geq 0$.

A továbblépésnél fontos figyelembe venni, hogy egy engedélyezett állapotátmenet nem feltétlen következik be azonnal. Amikor az óra értéke $cl \geq 2$, a B→F átmenet engedélyezett lesz, azonban az is elképzelhető, hogy a rendszer tovább várakozik B állapotban, átmenet csak később zajlik le és így a B→C átmenet fog bekövetkezni. Ha ez utóbbi történik, akkor C állapot lesz az aktuális és az óra értéke: $cl \geq 3$. Az óra pontos értékét azonban nem tudhatjuk, hiszen az állapotátmenet $cl = 4$ és $cl = 40$ értéknél egyaránt bekövetkezhetett.

A C állapotból újra két lehetséges folytatás van. Ha az óra értéke $3 \leq cl < 5$, akkor végrehajtódhat a C→D állapotátmenet. Azonban az is lehetséges, hogy $cl = 5$ eléréséig ez nem következik be és így a rendszer holtpontra kerül, C-ből nem léphet át más állapotba.

Ha a C→D állapotátmenet lezajlott és D állapotban tartózkodik a rendszer, akkor újra csak annyit tudhatunk az óra értékéről, hogy $cl \geq 3$. Hiába biztos, hogy az állapotátmenet idején `cl` értéke 3 és 5 közé esett, mivel D állapotban tetszőlegesen sok időt eltölthettünk, így `cl` értékére felső korlátot már nem tehetünk.



12. ábra Példa az óraváltozók működésére

2.4 Szinkronizáció

Ahogyan azt a bevetőben is írtuk, az UPPAAL időzített automaták hálózatát képes modellezni és verifikálni. Leegyszerűsítve ez azt jelenti, hogy több folyamatot is definiálhatunk, melyeket egy-egy állapotgép fog leírni. Azonban ezek a folyamatok nem csak egymástól függetlenül működhetnek, lehetséges köztük interakció. Erre szolgál az UPPAAL-ban a *szinkronizáció* művelet.

A szinkronizációnak két típusa van UPPAAL-ban: *egyszerű szinkronizáció* és *broadcast szinkronizáció*. Mindkettőhöz először egy *csatornát* kell definiálnunk, ezen keresztül fog megvalósulni a folyamatok közti üzenetküldés.

Egy egyszerű szinkronizációt támogató `ch` nevű csatornát a következőképp definiálhatunk:

```
chan ch;
```

Broadcast csatorna definiálásához a broadcast kulcsszót is használnunk kell:

```
broadcast chan bch;
```

Ez után már a csatornán küldés és fogadás műveleteket hozzárendelhetjük egy-egy élhez (állapotátmenethez). A `ch` csatornán küldést `ch!`, míg a `ch` csatornán fogadást `ch?` jelöli.

Egy egyszerű ch csatornán pontosan akkor lehetséges szinkronizáció, ha van egy olyan folyamat, amelynek aktuális állapotából létezik végrehajtható állapotátmenet, amelyre `ch!` van írva, illetve létezik egy másik folyamat is, amelynek van végrehajtható, `ch?`-val ellátott állapotátmenete. Ilyet mutat a 13. ábra. A keretek jelölik az egyes folyamatokhoz tartozó állapotokat. A piros színnel

jelzett állapotok az egyes folyamatok aktuális állapotai. Mivel az egyik folyamatban lehetséges a csatornán üzenet küldése, egy másik folyamatban pedig az üzenet fogadása, ezért a szinkronizáció megtörténik. Ha azonban a küldéshez nem lenne egy megfelelő fogadás vagy a fogadáshoz nem lenne megfelelő küldés, a szinkronizáció nem hajtódna végre. Ilyenkor a szinkronizációval ellátott állapotátmenet sem hajtható végre.

Amennyiben egy küldőhöz több fogadó is lehetséges, csak egy (véletlenszerűen kiválasztott) fogadóval zajlik le a szinkronizáció.

13. ábra Példa egyszerű szinkronizációra

Ezzel szemben a broadcast szinkronizáció egy küldő és több (0..*) fogadó közt zajlik le. A fogadó fél szempontjából nincs különbség a két szinkronizáció közt, azonban a küldő akkor is végrehajthatja az átmenetet, ha több fogadó is lehetséges, ilyenkor mindegyikben végbemegy az állapotátmenet.

3 Verifikáció UPPAAL-ban

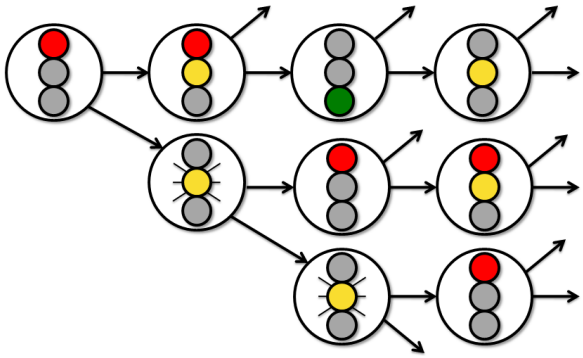
A modellezésen és szimuláción túl UPPAAL-ban a modellek verifikációjára is van lehetőség. Amennyiben a rendszerrel kapcsolatos követelményeinket formalizáljuk, ezek teljesülését vagy meghiúsulását vizsgálhatjuk.

A követelmények formalizmusaként az UPPAAL az ún. *CTL* (Computation Tree Logic) temporális logikai nyelv egy részhalmazát használja. Ennek segítségével bizonyos feltételek időbeli változásait vizsgálhatjuk.

Minden UPPAAL-beli CTL kifejezés felépítése a következő: «operátor» «logikai kifejezés». Logikai kifejezések lehetnek pl. változókra, óraváltozókra tett feltételek, pl. money==10000, cl>0, illetve állapotokkal kapcsolatos kifejezések, pl. d1.Working. Az operátor a következők egyike lehet: A[], A<>, E[], E<>. Ezek jelentésének megértéséhez meg kell ismerni a *számítási fát*.

A számítási fában a csúcsok (csomópontok) állapotokat reprezentálnak, az irányított élek pedig lehetséges állapotátmeneteket. A fa felépítéséhez először vegyük a teljes rendszer kezdőállapotát, mint v csúcsot. Keressük meg az összes olyan állapotot, mely elérhető a v állapotból, ezeket jelöljék a w_i csúcsok. Ez után húzzuk be az összes $v \rightarrow w_i$ élet, majd folytassuk ezt a keresést minden w_i-re [2]. Megjegyzendő, hogy ha egy x állapot elérhető a v és w állapotból is, akkor x-hez (legalább) két csúcsot is rendelni fogunk a fában, a v-ből és w-ből mutató élek nem ugyanabba a csúcsba fognak mutatni. Fontos azt is megjegyezni, hogy az esetek többségében ez a „fa” végtelen mélységű lesz.

Tekintsük példaképp egy közúti jelzőlámpa állapotait. Egy lámpa piros állapotból kerülhet piros-sárga állapotba, innen zöldbe, majd sárgába, végül újra pirosba. Minden állapotból továbbá átmehet villogó sárga állapotba. Villogó sárga állapotból csak piros állapotba térhet vissza. Az ehhez tartozó számítási fa részletét mutatja a 14. ábra.

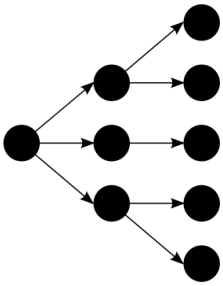
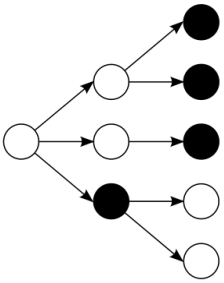
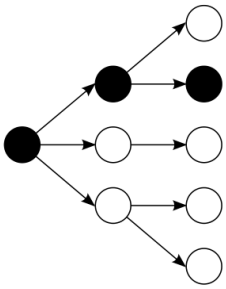
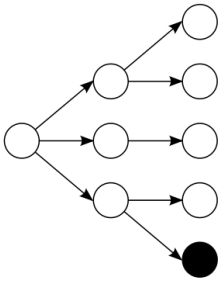


14. ábra Számítási fa részlete egy közúti jelzőlámpához

Ez után már definiálhatjuk az egyes operátorok jelentését:

- A[] «logikai kifejezés»: igaz, ha a számítási fa minden csúcsára igaz a kifejezés.
- A<> «logikai kifejezés»: igaz, ha a számítási fa minden útvonalán[2] legalább egy csúcsára igaz a kifejezés.
- E[] «logikai kifejezés»: igaz, ha a számítási fa legalább egy útvonalán minden csúcsra igaz a kifejezés.
- E<> «logikai kifejezés»: igaz, ha a számítási fa legalább egy csúcsára igaz a kifejezés.

Az operátorok intuitív jelentésének megértését segíti az 1. táblázat. A sötéttel jelölt állapotok azok az állapotok, amelyre a «logikai kifejezés» teljesül. Az egyes ábrák olyan számítási fákat mutatnak, amelyekre az adott kifejezés teljesül.

A[] «kifejezés»	A<> «kifejezés»	E[] «kifejezés»	E<> «kifejezés»
			

1. táblázat Az UPPAAL CTL-operátorainak intuitív jelentése

Van továbbá két speciális operátor is:

- A[] not deadlock: igaz, ha nincs holtpont a rendszerben.
- «logikai kifejezés 1» --> «logikai kifejezés 2»: igaz, ha az 1. logikai kifejezés bekövetkezése után mindig következik előbb vagy utóbb minden útvonalon legalább egy állapot, amelyre a 2. logikai kifejezés igaz

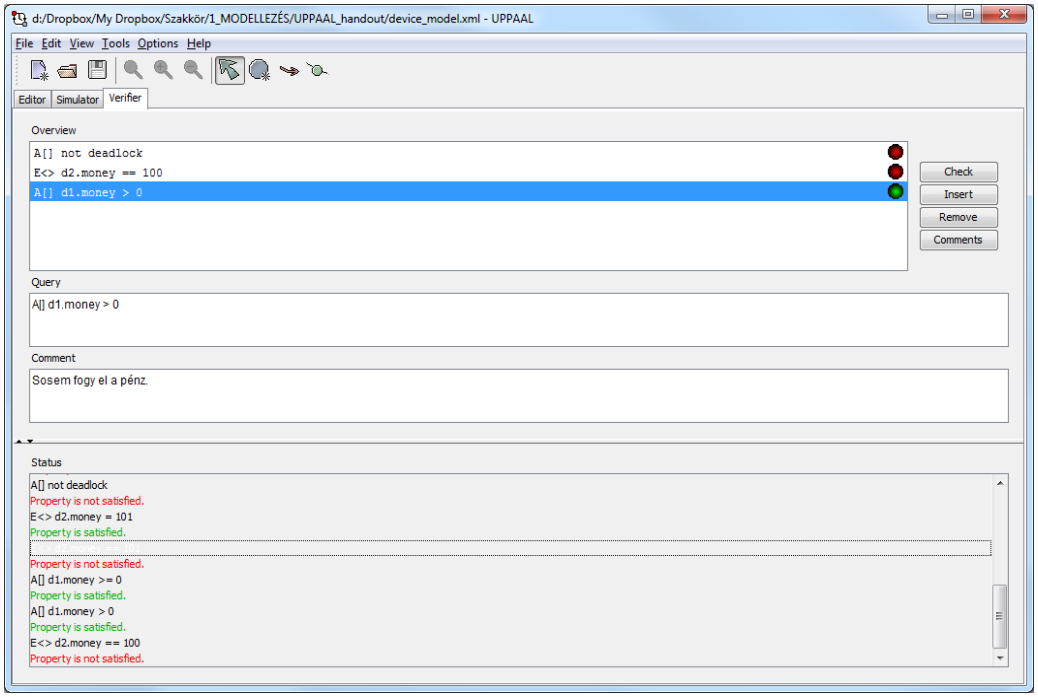
Ezek alapján például érvényes kifejezés a következő: A[] d1.money>=1000. Ez a kifejezés pontosan akkor igaz, ha a rendszerben a *d1* példány *money* változó értéke sohasem kisebb 1000-nél.

Megjegyzendő, hogy az egyes kifejezések nem függetlenek egymástól, igazak az alábbi összefüggések [4]:

[2] Valójában ez pongyola megfogalmazás. Csak azon (irányított) utakra kell teljesülnie az állításnak, amelyek a fa gyökeréből indulnak és végtelen hosszúak vagy olyan csúcsban végződnek, amelyből nem mutat ki él.

- ¬(A[]«logikai kifejezés») = E<> ¬«logikai kifejezés»
- ¬(A<>«logikai kifejezés») = E[] ¬«logikai kifejezés»

Az UPPAAL-ban ilyen kifejezések ellenőrzésére a *Verifier* használható. Ezt mutatja a 15. ábra. Itt a *Query* mezőben megadható egy temporális logikai kifejezés, amely *Check* gomb megnyomásával ellenőrizhető. A kifejezés mellett lévő kör színe jelzi, hogy teljesül-e a kifejezés (piros – nem teljesül, zöld – teljesül, szürke – nincs kiértékelve).



15. ábra UPPAAL Verifier

Ha az *Options* menüben a *Diagnostic trace* beállítást *None*-ról átállítjuk (pl. *Shortest*-re), akkor a kifejezések kiértékelése során lehetőségünk van egy példa vagy ellenpélda megtekintésére a *Simulator* nézetben. Így könnyebb az esetleges hibák felderítése és javítása.

A logikai kifejezések szintaktikáját itt nem tárgyaljuk. Precíz definíciója megtalálható a [4] dokumentumban.

Hivatkozások

[1] Gerd Behrmann, Alexandre David, and Kim G. Larse: A Tutorial on Uppaal 4.0
http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

[2] Formális módszerek tárgy segédanyagai
https://www.inf.mit.bme.hu/edu/courses/form/materials

[3] UPPAAL2k: Small Tutorial
http://www.it.uu.se/research/group/darts/uppaal/tutorial.pdf

[4] Gerd Behrmann: Introduction to UPPAAL
http://people.cs.aau.dk/~srba/courses/SV-05/slides/l11.pdf