Formális Módszerek (VIMIMA07)

Házi Feladat dokumentáció

# Formális modell

A megalkotott modell teljes egészében (részleteiben a dokumentáció további szekciói taglalják):

*Globális deklarációk:*

**const** int N = 3;  
**const** int B = 6;  
int[0,N-1] getter;  
broadcast chan request;  
urgent chan data[N];  
int[0,1] measurements[N];

*Processz-lokális deklarációk:*

int[0,B] battery = B;  
clock timer;  
int[-1,N-1] counter = -1;  
int[0,1] measurement;

*Sensor (parameter: int[0,N-1] pid):*

## 

# Tervezési lépések

Ez a szekció leírja a tervezési lépéseket ami az informális leírástól a formális modell megalkotásáig vezetett.

## Követelmények

Az informális leírásból a következő (szintén informális, de strukturált) listát állítottam össze a követelményekről:

1. *A szenzorhálózat N számú szenzorból áll.*
   1. *Pontosan 1 szenzor kérő típusú egy adott pillanatban*
      1. *A szenzorok váltják ezt a szerepet egymás között: 0, 1, .. N-1, 0*
   2. *Aki nem kérő típusú, az küldő típusú szenzor*
2. *A szenzorhálózat szenzorai tevékenységek között várakozás nélkül váltanak.*
   1. *Egy szenzor töltheti az akkumulátorát.*
      1. *A szenzor addig tölti az akkumulátorát, amíg tele nem lesz.*
      2. *1 egységnyi töltést 1 időegység alatt tölt vissza a szenzor.*
      3. *Az akkumulátor teljes feltöltése után alapállapotba*
   2. *Egy kérő típusú szenzor kérhet jelentést.*
      1. *Egy kérő típusú szenzor broadcast csatornán értesítés küld a jelentéskérésről.*
      2. *A broadcast értesítés azonnal megtörténik.*
      3. *A broadcast értesítés 2 egységnyi töltést fogyaszt.*
      4. *Egy kérő típusú szenzor dedikált csatornákon fogadja az adatot.*
      5. *A válaszokra összesen 1 időegységet vár a szenzor.*
      6. *Egy válasz feldolgozása azonnal megtörténik.*
      7. *Egy válasz feldolgozása 1 egységnyi töltést fogyaszt.*
      8. *A beérkezett jelentések számát a kérő szenzor eltárolja.*
      9. *A válaszok feldolgozása után alapállapotba visszatér (tehát újra tevékenységet választ)*
   3. *Egy küldő típusú szenzor mérhet, és küldhet jelentést, ha kap kérést.*
      1. *A mérés 1 időegység alatt történik meg.*
      2. *A mérés 1 egység töltést fogyaszt.*
      3. *A mérés egy véletlen szám 0 és 1 közül.*
      4. *Egy küldő típusú szenzor kaphat jelentéskérést.*
      5. *A jelentéskérés 1 egység töltést fogyaszt.*
      6. *A jelentéskérés azonnal megtörténik.*
      7. *A jelentéskérésre 1 időegységet vár a szenzor.*
      8. *A jelentéskérés után elküldi a mért adatot dedikált csatornán a kérő szenzornak.*
         1. *A jelentésküldés azonnal megtörténik.*
         2. *A jelentésküldés 1 egység töltést fogyaszt.*
      9. *A mért adat elküldése, vagy az időkeret letelte után alapállapotba visszatér*
3. *Mindegyik szenzor rendelkezik B kapacitású akkumulátorral.*
   1. *Egy tevékenységhez csak akkor kezdhet hozzá egy szenzor, ha elég a töltöttsége.*
   2. *Ha nem tud jelentéskérést fogadni a mérés után, a szenzor töltési tevékenységbe kezd.*
   3. *Akkor kezdhet egy szenzor tölteni, ha nincs teljesen feltöltve az akkumulátora.*

### Követelmények kiegészítése

E-mail konzultáció alapján a következőknek is teljesülnie kell:

1. *Szerepeljen az adatátadás a modellben is.*
2. *Az akkumulátor feltöltése után a szenzor rögtön visszatér alapállapotba.*
3. *Akkor kezdhet adatot kérni a kérő szenzor, ha töltöttsége alapján minden szenzortól tud választ is fogadni.*
4. *Akkor kezdhet mérni a szenzor, ha töltöttsége alapján azt el is tudja sikeresen végezni (nem kell feltétlen tudnia elküldeni az adatot).*
5. *Akkor fogadhat jelentéskérést mérés után a szenzor, ha töltöttsége alapján fogadni, majd adatot küldeni is tud.*

## Követelmények megvalósítása

Annak bemutatására, hogy az általam megalkotott modell megfelel az elvárásoknak, bemutatom, mely részei valósítják meg az adott követelményt.

1. *A szenzorhálózat N számú szenzorból áll.*

Szabad paraméterként tartalmazza a *pid* értéket minden szenzor a [0;N-1] tartományon.  
Mivel a rendszer példányosításakor nem adok meg ennek értéket, minden lehetőséget példányosítani fog az eszköz – ezzel N darab szenzor jön létre. Az N egy globális konstans.

* 1. *Pontosan 1 szenzor kérő típusú egy adott pillanatban*

A kérő szenzor szerepét egy globális *int[0,N-1]* *getter* változó tárolja. Ahhoz, hogy egy szenzor eldönthesse, hogy ő éppen kérő típusú-e, a saját *pid* értékét kell ezzel összehasonlítania – ha egyezik, kérő, ha nem egyezik, küldő. Ezzel mindig 1 kérő és N-1 küldő lesz.

* + 1. *A szenzorok váltják ezt a szerepet egymás között: 0, 1, .. N-1, 0*

Amikor egy szenzor befejezi az adatkérést és -feldolgozást, visszatér az alapállapotba, és átállítja a *getter* értékét:  
getter = getter == N-1 ? 0 : getter+1  
Ezzel amennyiben *N-1* az értéke (tehát a legmagasabb *pid* értékű szenzor volt az aktuális kérő), 0-ra állítja, egyébként 1-el növeli. Eredetileg jól beállított értékkészletű változó túlcsordulásával akartam megoldani, azonban az UPPAAL eszköz nem így kezeli a változókat, ezért hibát okozott amikor „túlcsordult”.

* 1. *Aki nem kérő típusú, az küldő típusú szenzor*

Ld. **1.1.** – mindig N-1 küldő szenzor lesz.

1. *A szenzorhálózat szenzorai tevékenységek között várakozás nélkül váltanak.*

Az alapállapot (*Init*) „urgent”, tehát nem telhet idő benne. Ez azt eredményezi, hogy nincs üresjáraton egy szenzor sem – amennyiben ezt valami kikényszerítené, az hiba (ilyen eset pl.: B == N esetén soha nincs elég töltöttség a kérő szenzornak a kérés elvégzéséhez, viszont amennyiben tele van az akkumulátora, töltésbe se tud kezdeni – ez viszont pont olyan helyzet, amit nem szeretnénk lekezelni, sőt, jó ha ezt az eszköz hibásnak jelzi).

* 1. *Egy szenzor töltheti az akkumulátorát.*

Létezik átmenet az alapállapotból (*Init*) a töltési állapotba (*Charging*).

* + 1. *A szenzor addig tölti az akkumulátorát, amíg tele nem lesz.*

A *Charging* állapotból az *Init* állapotba egyetlen irányított út vezet, az *IsCharged* állapoton keresztül. Ez az átmenet azonban csak battery == B esetben engedélyezett.

* + 1. *1 egységnyi töltést 1 időegység alatt tölt vissza a szenzor.*

A *Charging* állapotba érkezéskor a *timer* nevű lokális óraváltozó értéke minden esetben 0 (*IsCharged*-ból is, és *Init*-ből is) a közvetlen átmeneteken. Ebben az állapotban addig tartózkodhatunk, amíg timer <= 1 az invariáns miatt, és akkor hagyhatjuk el az egyetlen kifele vezető átmeneten át, amikor timer >= 1 az őrfeltétel miatt. Ezzel a *Charging* állapotban biztosan 1 egységet fogunk tartózkodni.   
Ezen felül az egyetlen kivezető átmeneten nő 1 egységgel a töltöttség: battery++. Továbbá, a *Charging* állapotba való visszatérés egyből megtörténik, mivel az *IsCharged* állapot „urgent”. Tehát 1 időegység pontosan 1 töltöttség egységet tölt vissza.

* + 1. *Az akkumulátor teljes feltöltése után alapállapotba kerül.*

Az *IsCharged* („urgent”) állapotból csak akkor lehet az alapállapotba jutni, ha az akkumulátor teljesen fel van töltve. Továbbá amennyiben ez igaz, ez az egyetlen engedélyezett átmenet, és mivel a forrás „urgent”, muszáj késlekedés nélkül alapállapotba visszatérnie.

* 1. *Egy kérő típusú szenzor kérhet jelentést.*

Létezik átmenet az alapállapotból (*Init*) a kérő állapotba (*Requesting*).

* + 1. *Egy kérő típusú szenzor broadcast csatornán értesítést küld a jelentéskérésről.*

Az *Init* állapotból *Requesting* állapotba történő átmeneten szerepel a request! szinkronizációs kifejezés, ami a *request* broadcast csatornán történő üzenetküldést jelenti.

* + 1. *A broadcast értesítés azonnal megtörténik.*

Egy „urgent” állapotból (*Init*) lépünk át a *Requesting* állapotba, ami alatt nem megengedett az idő múlása.

* + 1. *A broadcast értesítés 2 egységnyi töltést fogyaszt.*

Az *Init* állapotból *Requesting* állapotba történő átmeneten szerepel a  
battery = battery - 2 kifejezés, ami az akkumulátor töltöttségéből 2 egységet elvesz.

* + 1. *Egy kérő típusú szenzor dedikált csatornákon fogadja az adatot.*

A *Requesting* állapotból induló hurokél reprezentálja az adat fogadását. A küldéssel való egyidejűséget a data channel tömb elemein történő szinkronizáció reprezentálja, az adatátadást a globálisan elérhető measurements: int[0,1] tömb elemeinek írása és olvasása reprezentálja. Az átmeneten egy sorsolás található (i : int[0, N-1]), aminek a nemdeterminizmusa miatt az eszköz minden lehetőséget meg fog próbálni – tehát olyan, mintha minden szenzor csatornáján egyszerre figyelne a kérő szenzor. Az adatot egy lokális változóba menti (measurement = measurements[i]), de ezt nem fogja felhasználni.

* + 1. *A válaszokra összesen 1 időegységet vár a szenzor.*

A *Requesting* állapotba érkezéskor (ez csak az *Init* állapotból lehetséges) a timer értékét nullázzuk, majd az állapoton belüli timer <= 1 invariáns és az *Init* állapotba tartó átmenet timer >= 1 őrfeltétele biztosítja, hogy pontosan 1 időegységet tartózkodjunk a *Requesting* állapotban, ezzel ennyi időt tölthessünk csak a válaszokra várással.

* + 1. *Egy válasz feldolgozása azonnal megtörténik.*

A *Requesting->Requesting* hurokél végrehajtása alatt nem telik el idő, mivel azonnal megtörténik amint tud a data[i] channel „urgent” tulajdonsága miatt.

* + 1. *Egy válasz feldolgozása 1 egységnyi töltést fogyaszt.*

Az adatfogadást reprezentáló élen a töltöttséget 1 egységgel csökkentjük: battery--

* + 1. *A beérkezett jelentések számát a kérő szenzor eltárolja.*

A lokális *counter* változóban számoljuk, hány kérés érkezett be. A *Requesting* állapotba való első belépéskor 0 az értéke (counter = 0 miatt), majd minden adatfogadással 1-el növeljük (counter++).

* + 1. *A válaszok feldolgozása után alapállapotba visszatér (tehát újra tevékenységet választ)*

Amint engedélyezetté válik a *Requesting->Init* átmenet a guard teljesülése miatt, meg is kell ezt lépnünk mivel a *Requesting* állapot invariáns kifejezése sértetté válna további várakozás hatására.

* 1. *Egy küldő típusú szenzor mérhet, és küldhet jelentést, ha kap kérést.*

Létezik átmenet az alapállapotból (*Init*) a *Measuring*, onnan a *WaitForRequest*, majd onnan a *ToSend* állapotba. Ezekkel (ebben a sorrendben) tud a szenzor **mérni**, **kapni kérést**, **küldeni adatot**.

* + 1. *A mérés 1 időegység alatt történik meg.*

A mérést a *Measuring* állapotban végzi el a szenzor, ahová érkezve a *timer* 0-ra van állítva, az invariáns miatt maximum 1 egységet várakozhat a szenzor, a kimenő átmenetek pedig legalább 1 egység után engedélyezettek. Tehát ebben az állapotban pontosan 1 időegységet tölt a szenzor.

* + 1. *A mérés 1 egység töltést fogyaszt.*

A *Measuring* állapotba mutató átmenet csökkenti 1-el az akkumulátor töltöttségét: battery--

* + 1. *A mérés egy véletlen szám 0 és 1 közül.*

A mérést a *Measuring* állapotba mutató átmenet szelekciós kifejezése (0 és 1 közül választ) valósítja meg: my\_measurement: int[0, 1]

* + 1. *Egy küldő típusú szenzor kaphat jelentéskérést.*

A mérés elvégzése után, amennyiben van elég töltöttsége az akkumulátornak, a szenzor átkerül a *WaitForRequest* állapotba, ahonnan elérhető a request csatornán történő üzenetfogadás (szinkronizáció): request?

* + 1. *A jelentéskérés 1 egység töltést fogyaszt.*

A *Measuring->WaitForRequest* átmeneten történik a jelentéskérés fogadása, és ugyanazon az élen szerepel az akkumulátor töltöttségének 1 egységgel való csökkentése: battery--

* + 1. *A jelentéskérés azonnal megtörténik.*

Mivel a jelentéskérés broadcast csatornán történik, azonnal lépnie kell a fogadónak, amint tud. Ezzel nem telhet el idő a *WaitForRequest->ToSend* átmenetek között.

* + 1. *A jelentéskérésre 1 időegységet vár a szenzor.*

A jelentéskérésre a *WaitForRequest* állapotban várakozik a szenzor, ahová érkezve a *timer* 0-ra van állítva, az invariáns miatt maximum 1 egységet várakozhat a szenzor, a kimenő átmenetek pedig legalább 1 egység után engedélyezettek. Tehát ebben az állapotban pontosan 1 időegységet tölt a szenzor.

* + 1. *A jelentéskérés után elküldi a mért adatot dedikált csatornán a kérő szenzornak.*

Amennyiben a *ToSend* állapotba léptünk, az egyetlen átmenet az *Init* állapotba vezet, amin beállítjuk a mért adatot a globális measurements tömb megfelelő elemén (measurements[pid] = measurement), illetve szinkronizálunk a megfelelő csatornán (data[pid]!). Mivel a „küldő” (! jellel) átmenet kifejezései hamarabb értékelődnek ki, mint a „fogadó”-é (? jellel), ezért a kérő szenzor mindig friss adatot kap.

* + - 1. *A jelentésküldés azonnal megtörténik*

Miután megkaptuk a jelentéskérést, az „urgent” *ToSend* állapotba kerülünk. Itt, az „urgent” jelző miatt nem telhet el idő, tehát azonnal elküldjük az adatot*.*

* + - 1. *A jelentésküldés 1 egység töltést fogyaszt.*

A *ToSend->Init* átmeneten csökkentjük az akkumulátor töltöttségét 1 egységgel: battery--

* + 1. *A mért adat elküldése, vagy az időkeret letelte után alapállapotba visszatér*

Amennyiben kaptunk jelentéskérést, azonnal megtörténik az adat elküldése és ezzel visszatérünk az alapállapotba (ld. **2.3.8.1.**). Amennyiben nem kaptunk 1 egység alatt jelentéskérést, a *WaitForRequest* állapot invariánsa 1 időegység után meg lenne sértve, ha nem haladnánk át az *Init* állapotba, mely átmenet csak 1 időegység után kerül engedélyezésre. Tehát pontosan 1 időegység után visszatér az alapállapotba a szenzor.

1. *Mindegyik szenzor rendelkezik B kapacitású akkumulátorral.*

Létezik egy B globális konstans, és minden szenzor akkumulátora erre inicializálódik (int[0,B] battery = B;). Ezt nem lehet meghaladni (ld. **2.1.3.**).

* 1. *Egy tevékenységhez csak akkor kezdhet hozzá egy szenzor, ha elég a töltöttsége.*

1. Csak akkor kérünk jelentést, ha mindet fogadni is tudjuk. (*Init->Requesting* átmenet feltétele, hogy a töltöttség legalább N-1+2=N+1 legyen, tehát magunkon kívűl mindenkitől tudjunk jelentést fogadni, és a jelentéskérést is el tudjuk küldeni).
2. Csak akkor fogadunk jelentést, ha legalább 1 egységnyire töltött az akkumulátor, tehát fogadni tudjuk (erre ebben a formában jelenleg nincs szükség, mivel ezt biztosítja a 1-es pont, de ha azt megváltoztatnánk, akkor itt hibába futhatnánk – és ártani nem árt).
3. Csak akkor kezdünk el mérni, ha legalább 1 egyésgnyire töltött az akkumulátor, tehát el tudjuk végezni.
4. Csak akkor fogadjuk a jelentéskérést, ha fogadni és adatot küldeni is tudunk, tehát a töltöttségnek legalább 2 egységnek kell lennie.
   1. *Ha nem tud jelentéskérést fogadni a mérés után, a szenzor töltési tevékenységbe kezd.*

A *Measuring* állapotból 1 időegység után *Charging* állapotba kerülünk, amennyiben a töltöttség nem engedi meg, hogy *WaitForRequest* állapotba kerüljünk.

* 1. *Akkor kezdhet egy szenzor tölteni, ha nincs teljesen feltöltve az akkumulátora.*

Amennyiben B>=2, ez teljesül, mert *Init* állapotból csak akkor kerülhetünk *Charging* állapotba, ha nem teljesen töltött az akkumulátor, és a *Measuring* állapotból csak akkor juthatunk *Charging* állapotba, ha az akkumulátor töltöttsége 0 vagy 1.

A további követelmények teljesülése:

1. *Szerepeljen az adatátadás a modellben is.*

Ld. **2.2.4.**

1. *Az akkumulátor feltöltése után a szenzor rögtön visszatér alapállapotba.*

Ld. **2.1.3.**

1. *Akkor kezdhet adatot kérni a kérő szenzor, ha töltöttsége alapján minden szenzortól tud választ is fogadni.*

Ld. **3.1.**

1. *Akkor kezdhet mérni a szenzor, ha töltöttsége alapján azt el is tudja sikeresen végezni (nem kell feltétlen tudnia elküldeni az adatot).*

Ld. **3.1.**

1. *Akkor fogadhat jelentéskérést mérés után a szenzor, ha töltöttsége alapján fogadni, majd adatot küldeni is tud.*

Ld. **3.1.**

*Megjegyzés: Ezek azért teljesülnek a fenti követelmények alapján automatikusan, mivel azoknál lehetett volna máshogy is dönteni, de a további követelmények alapján született a fent bemutatott megoldás.*

# Követelmények

A követelmények formalizálása, kiértékelése, és az eredmény magyarázata:

## Lehetséges, hogy egyetlen küldő szenzor sem válaszolt a kérő jelentéskérés üzenetére.

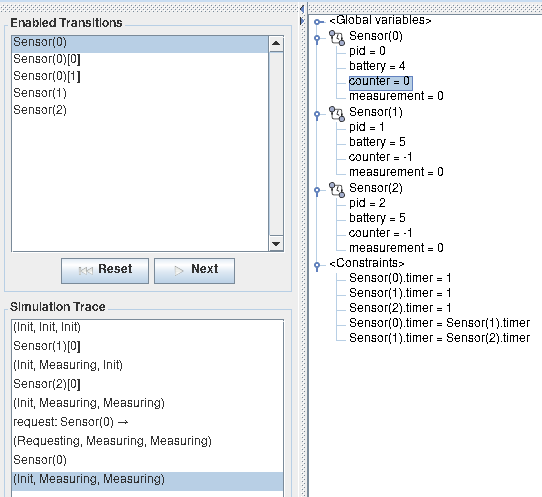
E<>exists(i : int[0,N-1])(Sensor(i).Init && Sensor(i).counter == 0)

A követelmény formalizálásának megkönnyítése érdekében a counter nem egy értelmezhető értékre, hanem -1 -re van inicializálva. Így az egyetlen lehetőség arra, hogy az Init állapotban a counter értéke 0 legyen (vagy akármi [0; N-1] között) az, hogy a legutóbbi kérés során ennyi válasz érkezett.

Maga a követelmény így hangzik formálisan:

*A bejárt útvonalak legalább egyikén legalább egy állapotban igaz, hogy legalább egy szenzor az init állapotban van, és a számlálója 0 értéket vesz fel.*

Az UPPAAL szerint ez **teljesül**. A következő példa trace-t adja:



Látszik, hogy a 0. szenzor counter-je 0, és init állapotban van. Előtte Requesting állapotban volt, de nem kapott egy választ sem.

## Lehetséges, hogy mindegyik küldő szenzor válaszolt a kérő jelentéskérés üzenetére.

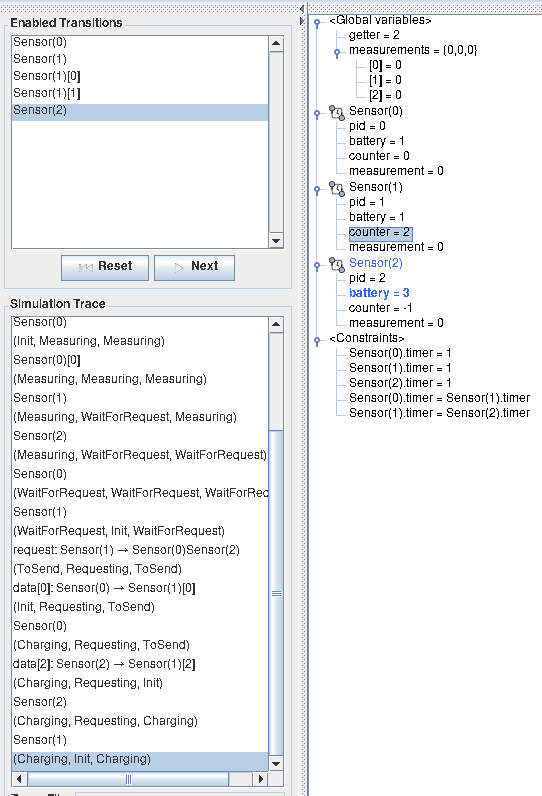
E<>exists(i : int[0,N-1])(Sensor(i).Init && Sensor(i).counter == N-1)

A követelmény formalizálásának megkönnyítése érdekében a counter nem egy értelmezhető értékre, hanem -1 -re van inicializálva. Így az egyetlen lehetőség arra, hogy az Init állapotban a counter értéke N-1 legyen (vagy akármi [0; N-1] között) az, hogy a legutóbbi kérés során ennyi válasz érkezett.

Maga a követelmény így hangzik formálisan:

*A bejárt útvonalak legalább egyikén legalább egy állapotban igaz, hogy legalább egy szenzor az init állapotban van, és a számlálója N-1 értéket vesz fel.*

Az UPPAAL szerint ez **teljesül**. A következő példa trace-t adja:



Látszik, hogy a 1. szenzor counter-je 2, és init állapotban van. Előtte Requesting állapotban volt, és 2 választ kapott.

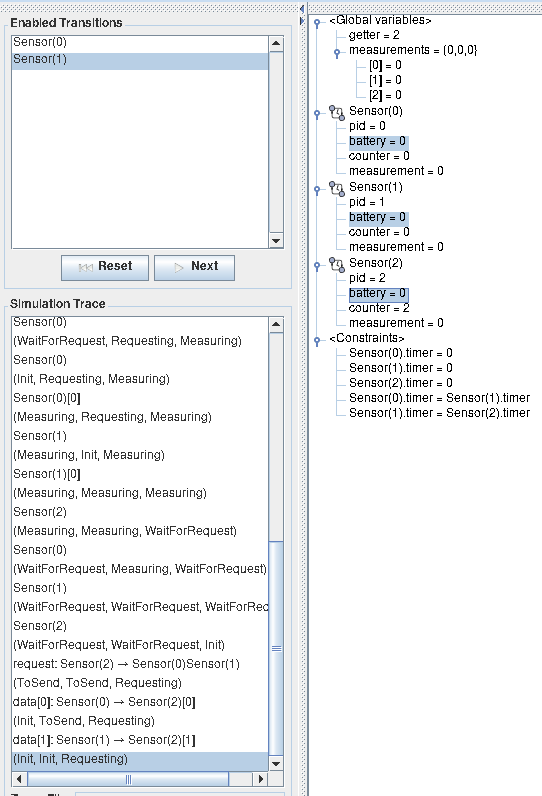
## Lehetséges, hogy minden szenzor egyszerre lemerül (mindegyik egyidőben 0 töltöttségi szintre kerül).

E<>forall(i : int[0,N-1])(Sensor(i).battery == 0)

A követelmény így hangzik formálisan:

*A bejárt útvonalak legalább egyikén legalább egy állapotban igaz, hogy mindegyik szenzor töltöttsége 0.*

Az UPPAAL szerint ez **teljesül**. A következő példa trace-t adja:



Látszik, hogy a mindegyik szenzor töltöttsége 0.