Állapottérképek

Előadásvázlat

Majzik István
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

2012. október 15.

Tartalomjegyzék

1. Bevezető

Az informatikai és villamosmérnöki tervezésben kiemelkedő fontosságúak az állapotdiagram alapú specifikációs, modellezési és tervező eljárások. Az állapotdiagramok (állapotgráfok) véges állapotterű rendszerek állapotainak, állapotátmeneteinek és a bemenő illetve kimenő eseményeknek (jeleknek) a reprezentációjára szolgálnak. Leírják, hogy egy adott állapotban tartózkodó rendszer adott bemeneti események hatására milyen kimenő eseményeket generál és milyen állapotba lép át.

Az állapotdiagramok széles körű használatát a következő kedvező tulajdonságai indokolják:

- Alkalmas mind hardver (pl. digitális vezérlők, processzorok) mind szoftver komponensek (pl. eljárások, objektumok) viselkedésének leírására.
- Intuitív, közel áll a mérnöki gondolkodásmódhoz.
- Egyszerű a grafikus reprezentáció. Legegyszerűbb esetben a grafikus nyelv elemkészlete az állapotokat jelölő körökből és az állapotátmeneteket jelölő, a ki- és bemenő eseményekkel címkézett nyilakból áll.
- Közvetlenül használható szimulációs célokra (a leírás „futtatható”) illetve kódgenerálásra.

Komplex rendszerek leírása esetén az előnyök mellett az egyszerű állapotdiagramoknak néhány hátránya is jelentkezik:

- Az állapotdiagram nem alkalmas konkurens rendszerek leírására. Ha ezek viselkedését pl. az állapotok minden lehetséges sorrendjét megadva szeretnénk leírni, az így előálló állapottér kezelhetetlenül nagy lehet, ugyanakkor elveszik a konkurens részrendszerek szinkronizációjának jól követhető leírása is.

- Hiányzik a hierarchikus finomítás lehetősége. Nem fejezhetjük ki, hogy állapotok egy csoportja összetartozik, pl. egy bemenő eseményre azonos módon reagál.
- Nem írható le a működés megszakítása majd a végrehajtáshoz való visszatérés.

Elsősorban ezen hátrányok indokolták az *állapottérkép* (statechart) alapú leírásmód kidolgozását. Az állapottérkép tehát az állapotdiagramok továbbfejlesztésével kialakított grafikus nyelv komplex, konkurens rendszerek leírásának támogatására. Az eredeti javaslatot [Har87] követően több változat („nyelvjárás") is kialakult, amelyek a lehetséges bővítések közül több-kevesebb használatát engedik meg.

Az állapottérképnek, mint grafikus nyelvnek, nincs szabványos szemantikája. Az egyes nyelvi elemek jelentése, az általuk meghatározott viselkedés elsősorban a leírt rendszertől, a célalkalmazástól és a támogató eszközöktől függ. Más-más szemantika lehetséges és célszerű, ha digitális vezérlők, valós idejű rendszerek vagy objektum-orientált szoftverek leírásáról és modellezéséről van szó. A [vdB94] cikk áttekintést ad sokféle kidolgozott szemantikáról. A leginkább elterjedt és támogatott állapotdiagram szemantikák a STATEMATE programcsomag által megvalósított szemantika [HN96] illetve az Unified Modeling Language (UML) által definiált szemantika.

A továbbiakban az UML állapottérképek elemkészletével és szemantikájával foglalkozunk, a STATEMATE szemantikára csak egy rövid összehasonlítás erejéig térünk ki.

2. Az UML állapottérkép elemkészlete

A következőkben az állapottérkép elemkészletét soroljuk fel. Minden elemhez megadjuk a grafikus reprezentációt és a használatra utaló intuitív jelentést (ez részben már a szemantikát előlegezi meg).

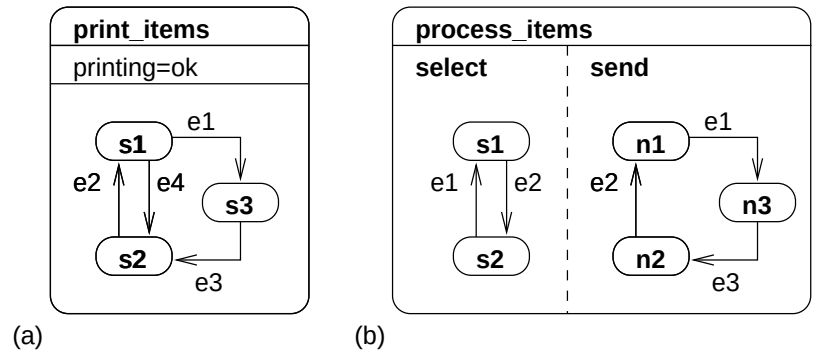
Állapot, szuperállapot, alállapot. A rendszer működése során különböző feltételeket teljesíthet, műveleteket végezhet vagy bekövetkező eseményeket várhat; ezeket a szituációkat a rendszer állapotainak nevezzük. Pl. adott feltételek teljesülése esetén azt mondjuk, hogy a rendszer az adott állapotban van; az állapotokat az (állapot)változók értéke, vagy az eddig elvégzett műveletek sorrendje határozhatja meg.

Az állapotdiagramokban egy állapot „oszthatatlan", míg az állapottérképek esetén egy állapotot finomítani lehet: az állapot által reprezentált feltételt vagy tevékenységet tovább finomítjuk, így *alállapotokat* definiálhatunk. A finomított állapot *szuperállapot* lesz, amely az alállapotok közös tulajdonságait fogja össze. Az alállapotokat, ugyanúgy, mint a szuperállapotokat, átmenetek kötik össze, így a szuperállapot „belsejében" egy újabb állapotdiagram, illetve további (rekurzív) finomítással állapottérkép alakítható ki.

Egy állapot grafikus reprezentációja egy lekerekített sarkú téglalap, ennek több tartománya lehet, amelyekbe az állapot neve, az (esetleges) állapotváltozók értéke, a belső akciók (ld. később) írhatók bele. Egy szuperállapot finomításával keletkező állapotdiagramot szintén a szuperállapotot reprezentáló téglalap egyik tartományába kell rajzolni (1.(a) ábra).

Konkurens állapot, régiók. Egy állapot adott műveletek végzését is reprezentálhatja. Ha ez a művelet konkurens résztevékenységekre osztható, akkor *konkurens állapotról* beszélhetünk. Az egyes résztevékenységekhez alállapotokat, úgynevezett *régiókat* rendelhetünk. Minden régió, mint szuperállapot, egy-egy állapottérképet tartalmazhat. Egy régió nem osztható közvetlenül újabb régiókra (azok az eredeti szuperállapot régióiként jelenhetnek meg).

A régiók grafikus reprezentációja úgy történik, hogy az állapotot jelképező lekerekített sarkú téglalapot szaggatott vonalakkal részekre osztjuk. A régiók finomításaként előálló állapotdiagramokat ezekbe a részekbe rajzoljuk (ld. 1.(b) ábra).



1. ábra. Állapotok megjelenítése

Összefoglalva tehát, egy (szuper)állapot finomítása a következők egyike lehet:

- VAGY típusú finomítás: az alállapotok egymást kölcsönösen kizárják, közülük egyszerre csak egy lehet aktív (és egynek aktívnak is kell lennie, ha a szuperállapot aktív).
- ÉS típusú finomítás: a finomított állapot konkurens; ha aktív, akkor egyszerre valamennyi alállapotának (régiójának) aktívnak kell lennie. Ez utóbbi azt jelenti, hogy minden régió VAGY típusú finomítása szükséges, és minden régióban az így előálló alállapotok közül egy aktív kell legyen.

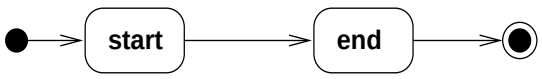
Ha egy állapot nincs tovább finomítva, akkor azt *egyszerű állapotnak* nevezzük. A finomított állapotot *összetett* vagy *szuperállapotnak* nevezzük. Egy szuperállapot illetve egy régió belsejében egy újabb állapotdiagram van, aminek állapotait tovább lehet finomítani, így rekurzívan újabb és újabb állapotdiagramokat felvenni. A (rekurzívan) finomított állapotokat egy diagramon, hierarchikus szerkezetben ábrázolva kapjuk a teljes állapottérképet, ami egyetlen csúcsszintű állapot finomításaként fogható fel.

Az állapothierarchia nemcsak az állapotfinomítást (top-down tervezés) segíti elő, hanem a megfelelő finomítási szint kiválasztásával és a részletek elrejtésével a leírás áttekinthetősége is növelhető.

Kezdőállapot és végállapot. Minden állapotdiagramban kijelölhető egy kezdő- és egy végállapot. Ha egy adott szuperállapotba illetve régióba lépünk, és az állapotátmenet nem jelöl ki közvetlenül egy célállapotot (tehát pl. csak a szuperállapot határához van az állapotátmenet húzva) akkor alapértelmezés szerint a kezdőállapotból kezdődik a működés.

A végállapotba lépés azt jelenti, hogy az állapotdiagram által leírt működés befejeződött, magasabb szinten a diagramot tartalmazó szuperállapot elhagyásával (majd esetleg oda újra belépve) folytatódhat a működés.

A kezdőállapot grafikus jelölése az állapotot reprezentáló lekerekített sarkú téglalaphoz egy külső, kitöltött körből (egy pszeudoállapotból) húzott él. A végállapotot az jelzi, hogy belőle egy él húzódik egy körből és egy benne lévő pontból álló pszeudoállapothoz (2. ábra).



2. ábra. Kezdõ- és végállapot megjelenítése

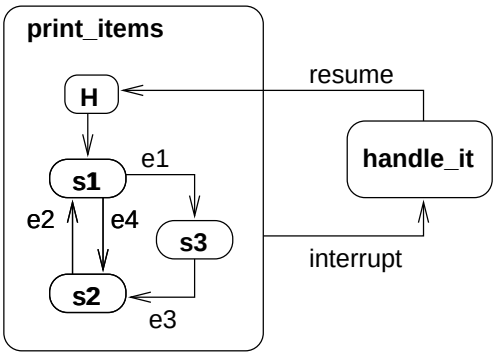
Emlékező és mélyen emlékező állapot. A rendszer viselkedését csak egyszerű állapotokkal leíró állapotdiagramban nem volt szükség annak leírására, mi történik az állapottér elhagyása esetén, hiszen az már kívül esett a leírás körén.

A hierarchikus állapottérkép esetén szükségessé válhat annak pontosítása, hogy mi történik egy szuperállapot „belsejében", ha a szuperállapotot (annak szintjén) elhagyjuk, illetve oda visszatérünk. Lehetséges, hogy a szuperállapot belsejét leíró állapottérképet elhagyva majd oda később visszatérve a kezdőállapotból kell a működésnek újraindulnia. De az is lehetséges, hogy az elhagyást megelőzően aktív állapotból kell folytatni a működést (mintha egy megszakítás érkezett volna, aminek kiszolgálása után az eredeti műveletek folytatódnak), tehát „emlékezik" a rendszer az utolsó aktív állapotra.

Az utóbbi eset leírására szolgálnak az *emlékező állapotok*. Egy szuperállapot illetve egy régió finomításával előálló állapotdiagramban legfeljebb egy emlékező állapot lehet. Ebből legfeljebb egy kimenő állapotátmenet indítható. Ha egy átmenet az emlékező állapotba lép, akkor ez valójában azt jelenti, hogy az állapotdiagram utoljára elhagyott aktív állapotába lép. Ha ez nem definiált (pl. az első belépés esetén), akkor az emlékező állapot kimenő átmenete által kijelölt állapotban folytatódik a működés.

Ha a fenti működésnek rekurzívan kell bekövetkeznie, tehát egy szuperállapot keretein belül a hierarchikus finomítással előálló valamennyi diagramban az utolsó aktív állapotban kell folytatni a működést, akkor ezt a legfelső szinten egy *mélyen emlékező állapot* jelzi.

Az emlékező állapot reprezentációja egy kör, benne H betű (3. ábra). A mélyen emlékező állapotot egy kör, benne H* jelzi.



3. ábra. Emlékezõ állapot megjelenítése

Állapotátmenet, összetett állapotátmenet, szintek közötti átmenet. A lehetséges állapotváltozásokat állapotátmenetek írják le. Egy egyszerű, egy szegmensből álló állapotátmenet két állapotot, egy kiindulási és egy célállapotot köt össze.

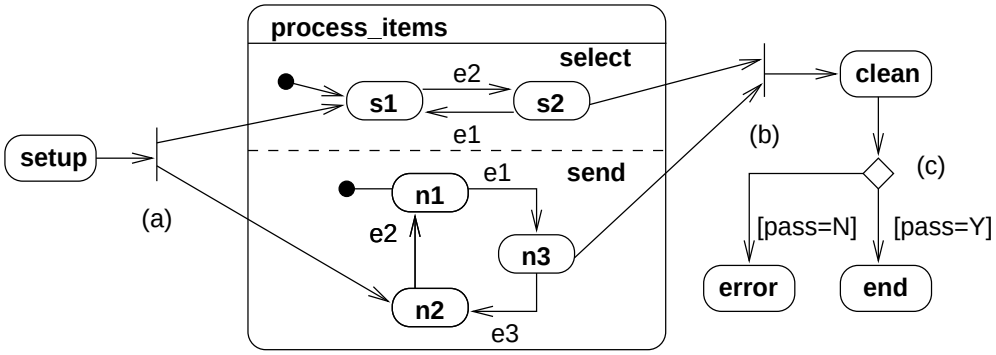
Több szegmensből álló, összetett állapotátmenetek következő formái lehetségesek:

- Szétváló átmenet. Ha a működés párhuzamos tevékenységekre bomlik, akkor az átmenetnek több célállapota lehet, amelyek egy szuperállapot finomításával előálló konkurens régiókban kell hogy legyenek.
- Egyesülő átmenet. Ha a működés párhuzamos tevékenységek szinkronizálása után folytatódik, akkor az átmenetnek több kiindulási állapota lehet, amelyek konkurens régiókból indulhatnak.

- Elágazó átmenet. Feltételektől függően az átmenet célállapota többféle lehet.

Az állapottérképes leírás megengedi, hogy legyenek állapotátmenetek a hierarchia eltérő szintjein lévő állapotok között is. Az ilyen, *szintek közötti átmenetek* jelentősen megnövelik a nyelv kifejezőképességét, hiszen egy, az állapothierarchia által nem korlátozott „goto” áll rendelkezésre (ugyanakkor problémákat okoznak a szemantika formalizálása terén).

Egy egyszerű állapotátmenetet egy irányított (nyílheggyel ellátott) él jelöl, a kiindulási állapotot reprezentáló szimbólumtól a célállapotot reprezentáló szimbólumhoz húzva. Egy összetett állapotátmenetet reprezentáló él több, irányított szegmensből áll, amelyek között pszeudoállapotok szimbólumai helyezkednek el. A szétválást (4. ábra, (a) részlet) illetve egyesülést (4. ábra, (b) részlet) függőleges vonalak, az elágazást egy rombusz reprezentálja (4. ábra, (c) részlet).



4. ábra. Összetett állapotátmenetek

Trigger események, feltételek és akciók. Egy állapotátmenet adott esemény hatására következhet be, ezt az eseményt *trigger eseménynek* nevezzük. A trigger esemény mellett az állapotátmenet bekövetkezése adott *feltételekhez* (pl. konkurens tevékenységek állapotához) is kötött lehet: az állapotátmenet csak akkor következhet be, ha a feltétel teljesül.

Az állapotátmenettel együtt a rendszer meghatározott tevékenységeket is végezhet (pl. változók beállítása), ezekre mint *akció szekvenciára* hivatkozunk.

A trigger események, feltételek és akciók az állapotátmenet címkéjeként szerepelnek:

```
trigger_event [guard] / action_sequence
```

Van néhány akció (tevékenység), amelyeknek speciális trigger eseménye van. Ezek az események a következők:

- `entry`: az állapotba való belépéskor (az állapot aktívvá válásakor) végrehajtandó akciók tartoznak hozzá.
- `exit`: az állapotból való kilépéskor (az állapot inaktívvá válásakor) végrehajtandó akciók tartoznak hozzá.
- `do`: az állapot aktív ideje alatt folyamatosan végrehajtandó akciók tartoznak hozzá. Ezek az akciók úgy is felfoghatók, mint az állapot finomításaként előálló konkurens állapotdiagramok nevei.

A fenti eseményeket az állapotot jelképező lekerekített sarkú téglalap egy tartományában kell felsorolni.

Belső átmenet. Az adott állapot elhagyása nélkül végrehajtandó akciókat *belső átmenetekhez* rendeljük. Ha ezek trigger eseménye teljesül, akkor az akciót végre kell hajtani, anélkül, hogy állapotváltozás történne (tehát sem a belépéshez, sem a kilépéshez tartozó akciókat nem kell végrehajtani; ebben különbözik egy belső átmenet egy olyan külső állapotátmenettől, amely ugyanabba az állapotba tér vissza).

A belső átmeneteket az állapotot jelképező lekerekített sarkú téglalap egy tartományában kell felsorolni.

Akció állapot ás befejező átmenet. Ha egy állapot elhagyását az állapothoz tartozó tevékenység befejeződése automatikusan elindítja, akkor azt az állapotot *akció állapotnak* nevezzük, az akció állapotból kiinduló állapotátmenetet pedig *befejező átmenetnek*.

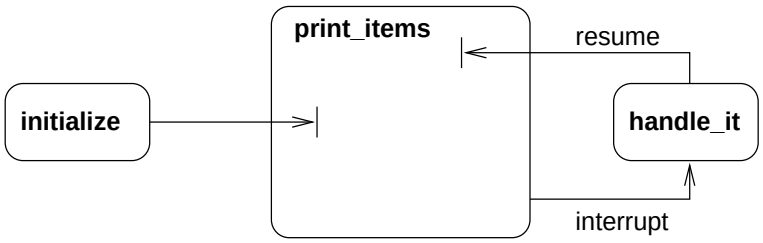
Az akció állapot grafikus reprezentációja egy egyenes alsó és felső határvonalakkal valamint convex oldalsó ívekkel határolt forma. A befejező átmenet jellegzetessége, hogy nincs trigger esemény illetve címke rajta.

Halasztott esemény. Ha egy adott eseményre egy adott állapotban nem kívánunk reagálni, de mégsem akarjuk, hogy az esemény elvesszen (u.i., mint később az UML szemantika leírásakor látni fogjuk, ha egy eseményre az állapottérképpel leírt állapotgép nem tud reagálni, akkor az az esemény elveszik), akkor az eseményre való reagálást halaszthatjuk. A halasztás egészen addig érvényes, míg olyan állapotba nem kerül a rendszer, ahol ez már nem előírt. Egy adott állapotban halasztott események listája az állapothoz van rendelve.

A halasztott eseményeket az állapotot jelképező lekerekített sarkú téglalap egy tartományában kell felsorolni.

Csonka állapotátmenet. Az állapottérkép segítségével lehetséges a részletek elrejtése is, tehát egy szuperállapotot finomító belső állapotdiagram elhagyása a grafikus reprezentációból. Ez esetben a szuperállapot belsejébe vezető átmenetek csonka átmenetek lesznek.

A grafikus reprezentációban a csonka átmenetek egy-egy függőleges vonalban záródnak az állapotot reprezentáló szimbólum belsejében (5. ábra).



5. ábra. Csonka állapotátmenet

3. Az UML állapottérkép informális szemantikája

Az állapottérkép szemantikája az állapottérképet „megvalósító” hipotetikus állapotgép viselkedésének (működésének) leírásával adható meg. Ezen állapotgép viselkedése a külső eseményekre való reagáláson, az állapotváltozásokon illetve a végrehajtott akciókon keresztül követhető.

Az állapotgép aktuális állapotát (amit a továbbiakban *konfigurációnak* fogunk nevezni) az állapothierarchia különböző szintjein található állapotdiagramok aktuális állapotai határozzák meg. Egy konfiguráció akkor érvényes, ha a következő feltételek fennállnak:

- A legfelső szintű (szuper)állapot aktív.
- Ha egy nem-konkurens szuperállapot aktív, akkor annak egy és csakis egy alállapota aktív.
- Ha egy konkurens állapot aktív, akkor annak minden régiójában egy és csakis egy alállapot aktív.
- Egy alállapot csak akkor lehet aktív, ha az azt tartalmazó szuperállapot aktív.

Az állapotgép viselkedésének szempontjából az UML szemantika két lényeges tulajdonsága emelhető ki:

- Az események feldolgozása egyenként, egymás után történik. Az állapotgép által generált és a környezetből érkező események egy (közelebbről nem definiált) eseménysorba kerülnek, ahonnan a (közelebbről szintén nem definiált) eseménysorrendező választja ki az állapotgép által feldolgozandó eseményt. Ezek a folyamatok az állapotgép szempontjából külső történések; az állapotgép szemantikája csak az eseményre való reagálást definiálja.
- Az események teljes feldolgozása („run-to-completion processing”). Ez azt jelenti, hogy (i) egy esemény hatására az állapotgépen belül az átmenetek maximális, konfliktusban nem lévő részhalmaza tüzel (aktivizálódik) valamint (ii) a következő esemény fogadása és feldolgozása addig nem történhet meg, amíg az aktuális esemény feldolgozása (az átmenetek tüzelése) be nem fejeződött, és az új stabil konfiguráció ki nem alakult.

Az alábbiakban végigkövetjük az állapotgép egy lépését, vagyis egy esemény hatására bekövetkező változásokat (az esemény feldolgozását).

- A kiválasztott esemény hatására az átmenetek egy része engedélyezetté válik. Pontosan azok az átmenetek lesznek engedélyezettek, amelyekre a következők állnak fenn:
  - minden kiindulási állapot az aktuális konfiguráció része;
  - a kiválasztott esemény az átmenet triggere;
  - van olyan útvonal a kiindulási állapotból a célállapotba, amelynek mentén minden feltétel ki van elégítve.

  A feltételek kiértékelése ebben a fázisban történik, még bármely állapotváltozás illetve akció bekövetkezése előtt. A feltételek kiértékelésének sorendje nem definiált, de minden feltétel mellékhatásmentes kell, hogy legyen.

  Az engedélyezett átmenetek száma és az esemény típusa alapján a következő (speciális) esetek különböztethetők meg:

  - Nincs engedélyezett átmenet, az esemény halasztott. Ebben az esetben az esemény függőben marad, egészen addig, amíg az aktuális konfiguráció (állapot) késlelteti. Ha az állapotgép olyan konfigurációba kerül, ahol a késleltetés megszűnik, akkor újra ki kell választani az eseményt és az állapotgépnek reagálni kell rá.
  - Nincs engedélyezett átmenet. az esemény nem halasztott. Ebben az esetben az esemény hatására változás nem történik, az eseményt el kell vetni.
  - Több engedélyezett átmenet van. Ebben az esetben közülük ki kell választani azokat, amelyek tüzelni fognak.

- A tüzelő átmenetek kiválasztása. Az átmenetek egy maximális halmaza fog tüzelni, amelyre a következők állnak fenn:

  – minden átmenet engedélyezett;
  – nincs konfliktus a halmazon belül (csak konfliktusban nem lévő, konkurens átmenetek tüzelhetnek egyidejűleg);
  – nincs a halmazon kívül olyan átmenet, amelynek prioritása magasabb lenne, mint egy, a halmazon belüli átmenet prioritása.

  Két (vagy több) átmenet akkor van konfliktusban, ha tüzeléskor az általuk elhagyott állapotok halmazainak metszete nem üres. A konfliktusok feloldása elsődlegesen a prioritások alapján történik; ha ez nem oldja fel a konfliktust, akkor a tüzelő átmenetet véletlenszerűen kell kiválasztani.

  Az átmenetek közötti prioritás a következőképpen definiálható:

  – Egy átmenet prioritását a kiindulási állapot határozza meg. Egyesülő átmenetek esetén a legalacsonyabb szintű (az állapotfinomítási hierarchiában a legmélyebben lévő) kiindulási állapot határoz meg.
  – Egy alállapotból kiinduló átmenet prioritása nagyobb, mint az azt tartalmazó szuperállapotból kiinduló átmenet prioritása.

- A kiválasztott átmenetek tüzelnek. A tüzelések sorrendje nem meghatározott (tetszőleges, véletlenszerű). Az állapotgép a kiválasztott állapotok tüzelésével egy új konfigurációba kerül. Ezt az állapotgép egy lépésének nevezzük.

  A konfiguráció változásának pontos meghatározásához az átmenetek következő jellemzőire lesz szükségünk:

  – Legalacsonyabb szintű közös ős (LCA) állapot: az az állapothierachiában legmélyebben található nem-konkurens állapot, amely az átmenetnek (illetve az átmenet egy kiválasztott útjának) mind az explicit forrásállapota(i)t, mind az explicit célállapota(i)t tartalmazza.
  – Legfőbb forrás: az LCA közvetlen alállapota, amely az explicit forrásállapoto(ka)t tartalmazza.
  – Legfőbb cél: az LCA közvetlen alállapota, amely az explicit célállapoto(ka)t tartalmazza.

  Egy átmenet tüzelését az alábbiakban tekintjük át.

- A legfőbb forrás állapotot és annak alállapotait elhagyja az állapotgép, majd végrehajtja a kilépési akciókat. Konkurens állapotok valamennyi alállapotát el kell hagyni. A kilépési akciókat az állapothierarchia szerint fordított sorrendben kell végrehajtani (először az alacsonyabb szintű állapotokhoz tartozó kilépési akciókat).

- Az átmeneten illetve annak kiválasztott útvonalán található akciókat végrehajtja az állapotgép (így általában új események generálása következik be). A végrehajtás sorrendje megfelel a szintaktikai sorrendnek illetve a szegmensek sorrendjének.

  Az akciók szinkronitása a következő lehet:

  – Szinkron akció: az átmenet végrehajtása addig befagy, amíg az akció végrehajtása be nem fejeződik.

  - Aszinkron akció: az átmenet végrehajtása nem fagy be.

- A legfőbb célállapotba lép az állapotgép, majd végrehajtja a belépési akciókat. A célállapot típusától függően rekurzívan be kell lépni annak alállapotaiba is, és a belépési akciókat ott is végrehajtani. A végrehajtási sorrend így megfelel az állapothierarchiának (a magasabb szintű állapotokhoz tartozó belépési akciókra kerül először sor).

  Az így rekurzívan aktívvá váló célállapotok következő eseteit különböztethetjük meg:

  - A célállapot nem konkurens. Ez esetben egyetlen alállapota lesz aktív:
    * Ha a tüzelő átmenet a célállapot határához csatlakozott, akkor ez az alállapot az alapértelmezett kezdőállapot lesz.
    * Ha a tüzelő átmenet belépett a célállapotba, akkor ez egyértelműen kijelöli az aktívvá váló alállapotot.
    * Ha a célállapotba belépő tüzelő átmenet egy emlékező állapotot jelöl ki, akkor az az alállapot lesz aktív, amelyik a legutóbbi kilépéskor aktív volt. (Ha most válik a célállapot először aktívvá, akkor az emlékező állapot kimenő átmenete jelöli ki az aktívvá váló állapotot.)
    * Ha a célállapotba belépő tüzelő átmenet egy mélyen emlékező állapotot jelöl ki, akkor az emlékező állapothoz képest annyi a különbség, hogy az alállapotokban *rekurzívan* a legutóbbi kilépéskor aktív állapotok válnak aktívvá.
  - A célállapot konkurens. Ez esetben minden régiójának egy-egy alállapota aktív lesz. Ha a tüzelő átmenet a célállapot határához csatlakozott, akkor ezek az alállapotok az alapértelmezett kezdőállapotok lesznek. Ha a tüzelő átmenet belépett egy régióba, akkor ez egyértelműen kijelöli az ott aktívvá váló alállapotot, és csak a többi régióban lesz az alapértelmezett kezdőállapot aktív.
  - A célállapotnak vannak befejező átmenetei. Ez esetben a célállapot nem stabil. Az eseménysorrendező befejező események sorozatát kell hogy továbbítsa az állapotgépnek, amíg az el nem éri az újabb stabil állapotot. (Elméletileg lehetséges, hogy a befejező átmenetek ciklikussága miatt soha nem érhető el stabil állapot. A gyakorlatban a befejező átmenetek számát korlátozni kell.)

A fenti szemantika algoritmikus leírása illetve megvalósítása az állapottérkép alulról felfelé történő bejárását igényli, hiszen egy (magasabb) szinten addig nem tüzelhet egy állapotátmenet, amíg ki nem derült, hogy alacsonyabb szinten nincs nála nagyobb prioritású engedélyezett átmenet.

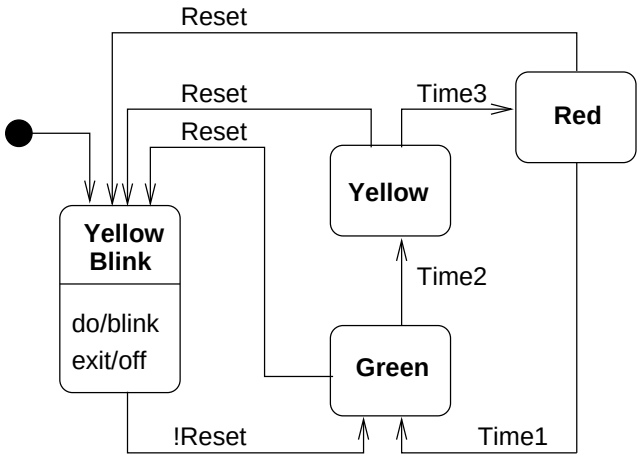
4. Egy példa

A példa során egy közúti kereszteződés forgalomirányító jelzőlámpa rendszerének vezérlőjét írjuk le. A vezérlő működése a következő (itt természetes nyelven megfogalmazott) feltételeknek kell, hogy megfeleljen:

- A jelzőlámpa rendszer egy fő- és egy mellékút kereszteződésében működik. A vezérlőtől elvárjuk, hogy soha ne mutasson zöldet egyszerre a két út forgalmának.
- A vezérlőt bármikor ki lehet kapcsolni, ilyenkor villogó sárgát mutat mindkét út irányába.
- Bekapcsoláskor a főút forgalmát kell először engedni. Az egy úton ellentétes irányba haladó autók áthaladása egyszerre történhet, a kanyarodó forgalom nem igényel külön irányítást.

- A két keresztező irány áthaladását a rendszer váltakozva engedi, a váltakozás ütemét egy időzítő szabályozza.
- Ha a főúton már legalább három autó várakozik tilos jelzésnél, akkor a rendszernek azonnal szabad jelzést kell biztosítania a számukra.
- Szintén a rendszer feladata, hogy a főúton tilos jelzés ellenére továbbhaladó autókat lefényképezze. Ezt a funkciót kézi vezérléssel lehet be- és kikapcsolni.

Az állapottérképes ábrázolás támogatja a vezérlő top-down tervezését, a megfelelő állapotok fokozatos finomításával. Az első lépésben a vezérlő négy állapotát különböztessük meg: a főút forgalmát engedő (`Green`), a mindkét úton sárga (`Yellow`), mellékút forgalmát engedő (`Red`) és a villogó sárga (`YellowBlink`) állapotot. Az így adódó állapottérképet az 6. ábra mutatja.



6. ábra. A finomítás elsõ lépése

Következő lépésként próbáljuk a ki- illetve bekapcsolást egyszerűbben leírni. A kikapcsolás bármely állapotból megtörténhet, így célszerű az eddigi (a normál működéshez tartozó) állapotokat egy szuperállapotba összefogni, és a kikapcsoláshoz tartozó állapotátmenetet innen indítani. Ezt az állapotátmenetet a `Reset` jel fogja triggerelni. (UML állapottérkép esetén az alállapotokhoz tartozó átmenetek nagyobb prioritásúak, mint a kikapcsolást jelentő fenti állapotátmenet, de mivel az események feldolgozása egyenként történik és konfliktus nincs, ez itt nem okoz működésbeli zavarokat.)

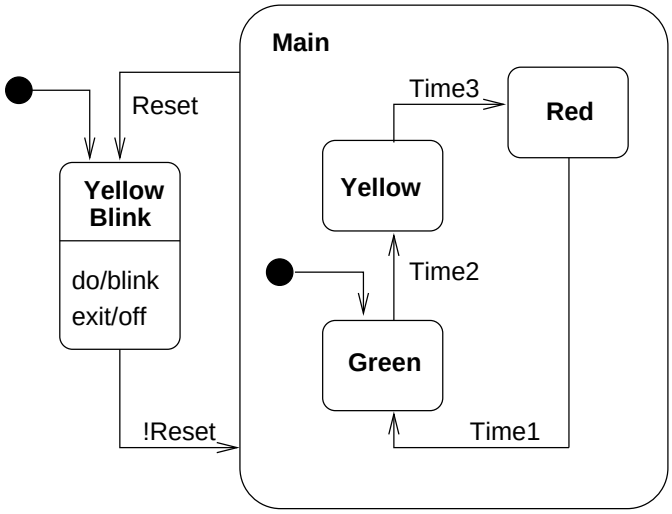
Bekapcsoláskor először a főút forgalmát kell a vezérlőnek elengednie, így a létrejövő szuperállapot kezdőállapota a `Green` kell legyen.

A finomítás után a 7. ábrán látható, már hierarchikus állapottérképet kapjuk.

Ha a főút lámpája piros, akkor a vezérlőnek két, párhuzamosan végzendő feladat van. Egyrészt számlálni kell a várakozó autókat, másrészt le kell fényképezni a pirosban áthaladókat (ez a funkció kikapcsolható). A két funkciót célszerűen két konkurens alállapot (régió) segítségével lehet leírni, amelyek a `Red` szuperállapotot finomítják. Ezek a `Camera` és a `Count` lesznek.

A fényképezést megvalósító `Camera` régiónak a kikapcsolhatóság biztosítása érdekében három alállapota lesz: a kikapcsolt állapot (`Off`), a bekapcsolt, pirosban áthajtó autóra várakozó állapot (`On`) és a fényképezést megvalósító állapot (`Shoot`). A fényképezés műveletének befejezése után automatikusan ismét várakozó állapotba kerül az alrendszer, tehát itt egy befejező átmenetre van szükség. Kikapcsolni csak várakozó állapotból lehet a fényképezést.

A főút lámpájának váltása nem szabad, hogy a fényképezés ki/bekapcsolt állapotát befolyásolja, így itt egy emlékező állapotra van szükség: pirosra váltás esetén abba az állapotba kell, hogy a fényképezés

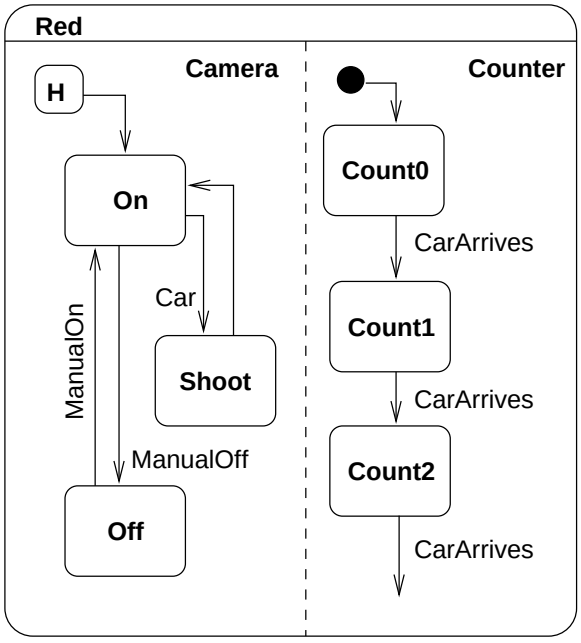


7. ábra. A finomítás második lépése

visszatérjen, ahol a zöldre váltáskor volt. Így a kezdőállapot jelölése helyett egy emlékező állapot kerül ebbe a régióba, amely első bekapcsoláskor engedélyezi a fényképezést.

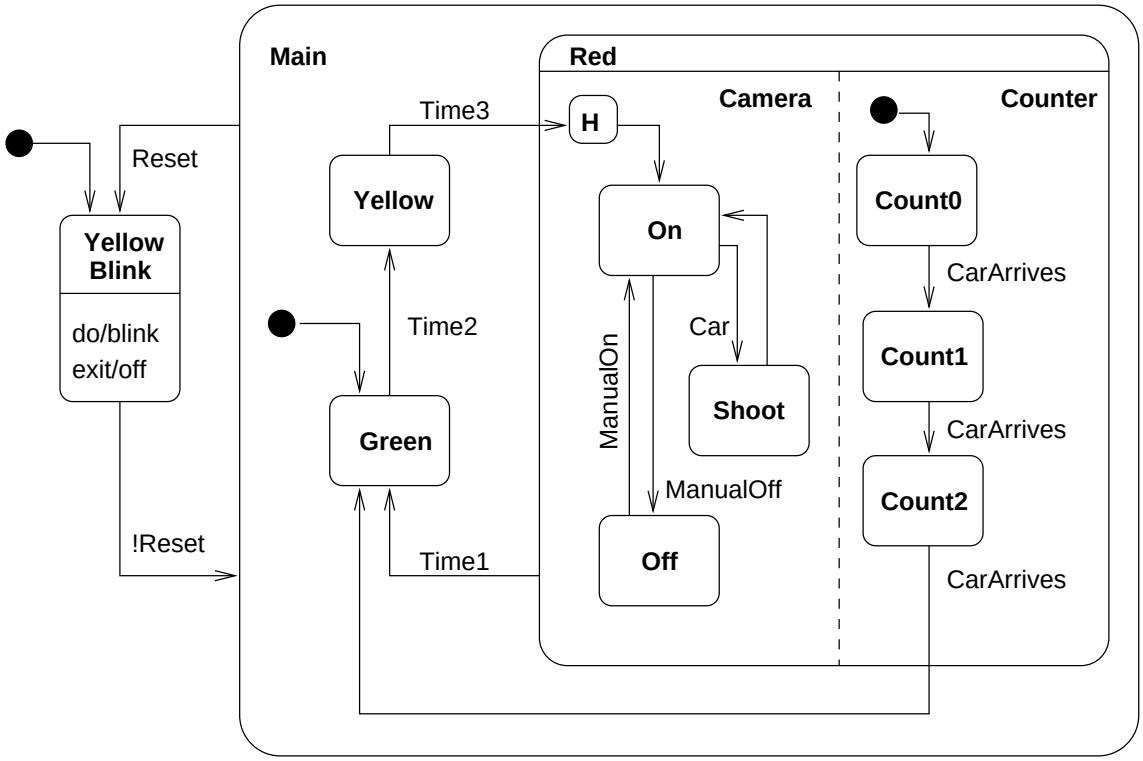
A számlálás ennél egyszerűbb funkció, nincs szükség félbeszakított működés folytatására. (Feltételezzük, hogy a piros jelzés idején várakozó autók áthaladnak a zöld jelzés alatt.) Az állapotok azt kódolják, hány autó várakozik már pirosban.

A 8. ábrán a `Red` állapot finomítása szerepel.



8. ábra. Egy állapot finomítása konkurens alállapotokkal

Végül a 9. ábra a vezérlő teljes állapottérképét mutatja be.



9. ábra. A vezérlõ teljes állapottérképes leírása

Hivatkozások

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[HN96] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[vdB94] M. von der Beek. A comparison of statechart variants. *Lecture Notes in Computer Science*, 863:128–148, 1994.