Eseményalapú rendszertervezés helyességbizonyítással

Majzik István
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
majzik@mit.bme.hu

2004

Bevezető

Cél: Modellalkotás és a modellfinomítás folyamatának bemutatása.

Alapkérdés:

- A finomítás során alkalmazott tervezői döntések valóban megfelelnek-e a kiindulási követelményeknek?
- A finomítás során létrejött modell teljes és ellentmondásmentes-e?

A módszer alapjai:

- A rendszertervezés *matematikai modellek* finomításával történik.
- Kezdeti modell: a feladat egy magas szintű, absztrakt megfogalmazása.
- Finomítási lépések: a feladat megoldásához szükséges szempontok.
- Részletes modell: az implementáció alapja.

A matematikai modellezés céljai

- Modellezés illetve a modellfinomítási lépések bizonyítottan helyes végrehajtása.
- Hibáktól mentes részletes modell az implementációhoz.
- (Futtatható kód automatikusan előállítása a formális modellből.)

A formális megközelítés nem helyettesíti a tervezői munkát:

- döntések meghozatala,
- a megoldás algoritmusának kidolgozása;
- "csak" a konzisztens finomítást ellenőrzi.

A modell elemei

- *Kontansok és változók*:
  - ⋆ Matematikai objektumok: pl. halmazok, relációk, függvények.
  - ⋆ Ezek reprezentálják a modellezett rendszer állapotát.
  - ⋆ *Invariánsok:* teljesítendő feltételek megkötése.
  - ⋆ Ez a modell statikus része.

- *Események*: Adott feltételek mellett megvalósuló állapotváltozás.
  - ⋆ Azt írják le, hogyan vesznek fel új értéket az egyes változók.
  - ⋆ *Feltételrész:* egy logikai igaz vagy hamis értéket felvevő kifejezés a konstansok és változók felhasználásával.
  - ⋆ *Akciórész:* a változók új értékének megadása az előző értékek felhasználásával.
  - ⋆ Ez a modell dinamikus része.

Esemény szemantika

- Esemény bekövetkezését nem kötjük külső kiváltó okhoz.
- Csak a következményt, a megfigyelhető hatást írjuk le.
- Az esemény akciórésze által leírt változás spontán módon bekövetkezhet, ha a feltételrész igaz.
- Az is lehetséges, hogy igaz feltételrész esetén sem következik be az akció által leírt változás.
- Az akció atomi, egyidejű értékadások halmaza (az itt fel nem tüntetett változók értéke nem módosul).

Külső nemdeterminizmus és szinkronitás

- Ha több esemény feltétele válik egyszerre igazzá,
  akkor közülük egyidejűleg csak egy esemény következhet be,
  ennek kiválasztódása véletlenszerű.

- Aszinkron végrehajtás: Az események aszinkron módon következnek be.

- Az egyedüli szinkronitás:
  az akciórész és ennek hatására más események feltételrészének igazzá válása.

- Konkrét eseménysorrend: explicit módon kell modellezni.

Szintaxis

Események megadása:

$$\begin{aligned}&\mathrm{Event1} \mathrel{\widehat{=}} \mathrm{ANY}\ x, y, ...\\&\mathrm{WHERE}\ P(x, y, ..., v, w, ...)\\&\mathrm{THEN}\ S(x, y, ..., v, w, ...)\ \mathrm{END}\end{aligned}$$

- Event1 az esemény neve,
- $x, y, ...$ lokális változók az esemény megadásához,
- $v, w, ...$ a modell változói illetve konstansai,
- $P(x, y, ..., v, w, ...)$ a feltétel,
- $S(x, y, ..., v, w, ...)$ az értékadás akciója.

A feltételrész jelentése

A feltételrész a következő egzisztenciális predikátum:

$$\exists(x, y, ...) : P(x, y, ..., v, w, ...)$$

Esemény bekövetkezése:

- Ha találunk az $x, y, ...$ lokális változókra olyan behelyettesítést, hogy $P(x, y, ..., v, w, ...)$ igazzá válik.

Belső nemdeterminizmus:

- A behelyettesítés megválasztásában érvényesülő szabadság.

Egyszerűbb alakok

Nincs lokális változó:

$$\text{Event1} \widehat{=} \text{SELECT } P(v, w, ...) \text{ THEN } S(v, w, ...) \text{ END}$$

Nincs feltételrész:

$$\text{Event1} \widehat{=} \text{BEGIN } S(v, w, ...) \text{ END}$$

Az $S(x, y, ..., v, w, ...)$ akciórész lista formában
(itt $E, F, ...$ a $v, w, ...$ változók új értékét megadó kifejezések):

$$v, w, ... := E, F, ...$$

Hosszabb listák esetén: Elválasztás az $\|$ operátorral:

$$\begin{aligned} v &:= E \,\| \\ w &:= F \,\| \\ &... \end{aligned}$$

Értékadás általánosított formája

$$v, w : R(v, w, v_0, w_0)$$

- v és w tetszőleges értéket vehetnek fel, amelyek kielégítik az $R(v, w, v_0, w_0)$ feltételt.
- v_0 és w_0 a változók előző értékei.
- Ilyen általános formában az előbbi $v = E(v, w)$ illetve $w = F(v, w)$ értékadás a következő:

$$v, w : (v, w = E(v_0, w_0), F(v_0, w_0))$$

A modellfinomítás

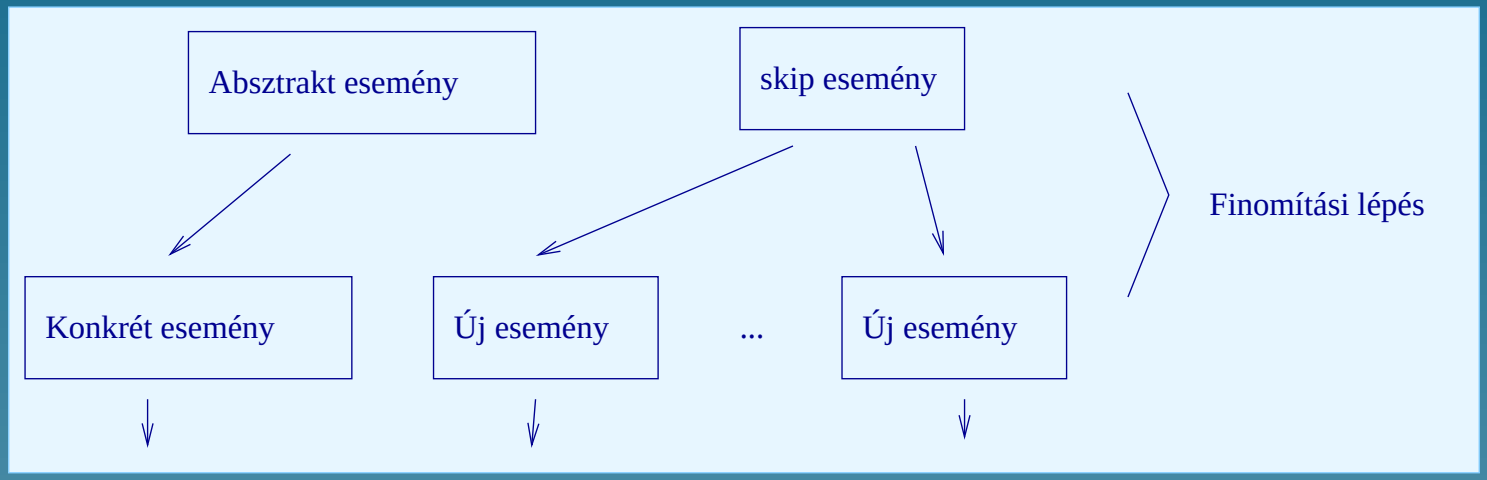
- Adott (absztrakt) modellből egy finomított (konkrét) modellt hozunk létre.
  - ⋆ Olyan állapot-részek illetve változások leírása, amelyeket az előző modell még nem tartalmazott.
  - ⋆ Így egyre több részletet vonhatunk be a modellbe.
- Az "absztrakt" és "konkrét" jelzők itt viszonylagosak.
- A finomítás során:
  - ⋆ Új változók (konstansok) bevezetése.
  - ⋆ Új események bevezetése.

Állapotok és események finomítása

- Az állapot finomítása:
  - ⋆ Meg kell adni azt az *invariánst*,
    ami összekapcsolja az absztrakt modell állapotát (változóit)
    és a konkrét modell állapotát (változóit).
  - ⋆ Pl. absztrakt modellben egy v változó,
    a konkrét modellben pedig egy w változó,
    akkor az invariáns $J(v, w)$ alakban adható meg.

- Események finomítása:
  - ⋆ Az absztrakt modell minden eseményéhez kell tartozzon
    egy (adott esetben módosított) esemény a konkrét modellben.
  - ⋆ Új eseményeket is felvehetünk
    (ezek formálisan az üres skip eseményt finomítják).

Események finomítása



A modellfinomítás helyességének bizonyítása

Cél:

- Az egész modellre előírt invariánsok betartása.
- A finomítási lépések konzisztenciája:
  a finomított modell és az absztraktabb modell közti megfelelőség.

Ezek megmutatásához 4 tulajdonságot kell bizonyítanunk:

- Az állapotokra vonatkozó invariánsok betartása.
- A modellfinomítás helyessége.
- A konkrét események feltételrészének megfelelő szigorítása.
- Új események kizárólagosságának elkerülése.

1. Az invariánsok betartása

- Minden eseményre be kell bizonyítanunk:
  betartja az állapotokra vonatkozó előírt invariánsokat,
  az akciórész úgy módosítja a változókat,
  hogy az invariánsok nem sérülnek.

- A bizonyítás alapja:

  ⋆ az esemény bekövetkezéséhez a feltételrésznek igaznak kellett lennie,
  ⋆ a változók előző értékei teljesítették az invariánsokat.

- Jelölés:

  ⋆ az állapotváltozó v,
  ⋆ az invariáns $I(v)$,
  ⋆ az esemény

$$\text{ANY } x \text{ WHERE } P(x, v) \text{ THEN } v' : R(x, v', v) \text{ END}$$

A bizonyítandó kifejezés

$$I(v) \wedge P(x, v) \wedge R(x, v', v) \Rightarrow I(v')$$

- Ha az akciórész az egyszerűbb $v = E(x, v)$ formában adott:

$$I(v) \wedge P(x, v) \Rightarrow I(E(x, v))$$

- Ha az esemény egyszerűen

$$\text{SELECT } P(v) \text{ THEN } \ v = E(v) \text{ END}$$

akkor a bizonyítandó kifejezés

$$I(v) \wedge P(v) \Rightarrow I(E(v))$$

2. A modellfinomítás helyessége

Be kell bizonyítanunk:

- A konkrét esemény feltételrésze szigorúbb,
  mint az absztrakt esemény feltételrésze,
  - ⋆ tehát a konkrét esemény kevésbé gyakran fordul elő,
    mint az absztrakt esemény
  - ⋆ (ld. még a következő bizonyítandó állításokat is).
- A konkrét esemény akciórészének hatása ugyanaz,
  mint az absztrakt esemény akciórészének a hatása
  - ⋆ de lehetőség szerint kevesebb belső nemdeterminizmussal.

Jelölés

- Az absztrakt modellben az állapotváltozó v,
  az invariáns $I(v)$,
  az esemény pedig a következő:

$$\text{ANY } x \text{ WHERE } P(x,v) \text{ THEN } v := E(x,v) \text{ END}$$

- A konkrét modellben az állapotváltozó w,
  az állapot finomításához tartozó invariáns $J(v,w)$,
  az esemény pedig a következő:

$$\text{ANY } y \text{ WHERE } Q(y,w) \text{ THEN } w := F(y,w) \text{ END}$$

A bizonyítandó állítás

$$I(v) \wedge J(v,w) \wedge Q(y,w) \Rightarrow \exists x : (P(x,v) \wedge J(E(x,v), F(y,w)))$$

Ez azt jelenti, hogy

- A konkrét esemény lokális változójának minden választása esetén van az absztrakt eseménynek olyan lokális változója, ami esetén az állapot finomításához tartozó invariáns igaz lesz (az események által módosított állapotváltozókra).

- A konkrét esemény feltételrésze szigorúbb, mint az absztrakt eseményé.

Egyszerűbb alakok

- Absztrakt esemény:

$$\text{SELECT} \;\; P(v) \;\text{THEN} \;\; v := E(v) \;\text{END}$$

- Konkrét esemény:

$$\text{SELECT} \;\; Q(w) \;\text{THEN} \;\; w := F(w) \;\text{END}$$

- A bizonyítandó állítás a következő:

$$I(v) \wedge J(v,w) \wedge Q(w) \Rightarrow (P(v) \wedge J(E(v), F(w)))$$

3. A konkrét események feltételrésze szigorításának korlátai

- Az előző pont előírása:
  - ⋆ Egy konkrét esemény feltételrészének szigorúbbnak kell lennie, mint az absztrakt esemény feltételrészének.
- A szigorítást a finomítás során felvett *új eseményeknek* kell kompenzálniuk:
  - ⋆ E nélkül az absztrakt modellben meglévő viselkedés "elveszne" a finomítás során.

Bizonyítandó állítás

- Az absztrakt esemény feltételrésze
  (amiről tehát láttuk, hogy gyengébb a konkrét esemény feltételrészénél)
  szigorúbb,
  mint a konkrét esemény és az ide tartozó új (a finomított modellben szereplő)
  események *feltételrészeinek diszjunkciója*.

- Ez azt jelenti: A konkrét esemény azért fordul elő kevésbé gyakran,
  mint az absztrakt esemény,
  mert a finomítás során felvett új események is előfordulhatnak.

- Ha a finomítás során nem veszünk fel új eseményeket:

  ⋆ Az absztrakt és a konkrét események feltételrészeinek meg kell egyezniük.

4. Új események kizárólagosságának elkerülése

Bizonyítani kell:

- Az újonnan felvett események nem gátolhatják meg minden esetben a konkrét esemény bekövetkezését.
- Ellenkező esetben a finomítás nem volna helyes, hiszen az absztrakt esemény még bekövetkezhetett.

A probléma jellemzői

- Független a konkrét események feltételrészének szigorításától.
- Itt a konkrét és az új események feltételrészeinek permanens átlapolódásáról van szó.
- Azt kell bebizonyítanunk, hogy ilyen permanens átlapolódás nem lép fel.
- Nem következik be a konkrét esemény "kiéheztetése".

Esemény alapú rendszertervezés mintapélda

- Egy absztrakt, magas szintű modellből kiindulva kell
  a feladat megoldásához szükséges szempontokat rendre figyelembe véve
  a részletes (konkrét) modellhez eljutni.

- Eseményvezérelt alapú modellfinomítás:

  ⋆ események kidolgozása,
  ⋆ a finomítás során ezek módosítása,
  ⋆ újabb események felvétele.

- Mintapélda: egy beléptető rendszer.

A beléptető rendszer mintapélda

- Feladata: Adott személyek adott termekbe való bejutását lehetővé tegye illetve meggátolja.
  - ⋆ Minden személynek van egy állandó jogosultsági listája.
  - ⋆ Adott terem ajtajánál a beléptető rendszer dönti el, hogy beengedhető-e az adott személy.
  - ⋆ A termekből való kilépés is jogosultsághoz kötött.
  - ⋆ Minden személy egy-egy mágneskártyával azonosítja magát az ajtónál.
  - ⋆ A termek ajtajai mint forgóajtók vannak kiképezve.
  - ⋆ Az ajtó mellett egy-egy zöld és piros lámpa található.
  - ⋆ Minden forgóajtó csak egyirányú áthaladást tesz lehetővé.
  - ⋆ A beléptetés folyamata: Azonosítás, lámpa, forgóajtó mozgatás.

Nem eldöntött kérdések

- A beléptetés központosított vagy minden forgóajtónál lokális döntéssel?
- Milyen forgóajtó illetve kártyaolvasó áll rendelkezésre?
  (Pl. automatikus blokkolás áthaladás után.)
- Hogyan garantálhatjuk,
  hogy egy személy nem marad bezárva valamelyik teremben?
- Milyen legyen a beléptető rendszer viselkedése extrém használat esetén?
  (Pl. újabb mágneskártya behelyezése miközben a zöld lámpa világít és az előző személy még nem haladt át.)

Ideális eset:

- Ezek a kérdések az absztrakt modell finomítása során felszínre kerülnek!
  (Akkor is, ha a tervező nem gondol rá, a bizonyítás a döntést "kikényszeríti".)

A kiindulási modell

Jelölés: P a tulajdonságok, D a döntések jele.

- *P1: A rendszer személyeket és termeket kezel.*
  - ⋆ Ezen állítás alapján felvehető két nem-üres halmaz:

$$prs \neq \emptyset$$
$$bld \neq \emptyset$$

- *P2: Minden személynek jogosultsága van adott termekbe belépni.*
  A jogosultságok nem változnak.
  - ⋆ A jogosultságok mint személy-terem párok vehetők fel (bináris reláció):

$$aut \in prs \leftrightarrow bld$$

- *P3: Minden személy egy adott pillanatban egy adott teremben tartózkodik.*

- *D1: A rendszer a személyek termek közötti mozgását kontrollálja.*
  - ⋆ A termek között folyosókat vagy külső teret nem tételezünk fel.
    Ez már döntés!

- *P4: Egy adott időpontban egy személy pontosan egy teremben tartózkodik.*
  - ⋆ Ez az előbbi döntésből adódik.
  - ⋆ Így a személyek tartózkodási helyét egy teljes függvény írja le.
    Ez lesz az állapotváltozó a modellben:

$$sit \in prs \rightarrow bld$$

- *P5: Egy adott teremben tartózkodó személynek van jogosultsága abban a teremben tartózkodni.*
  - ⋆ Ez a helyes működés feltétele.
  - ⋆ Invariáns: A tartózkodási helyet leíró függvény
    benne van a jogosultságot leíró relációban:

$$sit \subseteq aut$$

A változásokat leíró esemény

- Az egyes személyek mozgását (tartózkodási hely változását) határozza meg.
- Ha egy személynek van jogosultsága egy terembe belépni,
  és nem abban a teremben tartózkodik, akkor beléphet oda:

$$\text{pass} \mathrel{\widehat{=}} \text{ANY } p, b \text{ WHERE } (p, b) \in aut \wedge sit(p) \neq b \text{ THEN } sit(p) := b \text{ END}$$

- Ezen az absztrakciós szinten:
  ez az egy esemény írja le a rendszerben lezajló változásokat.
- Az eseményben leírt változás nem feltétlenül megy végbe,
  csak azt írtuk le, hogy bekövetkezhet!
- Könnyű bebizonyítani, hogy az esemény akciórésze során
  az invariáns igaz marad (ezt a feltételrész biztosítja).

Az első finomítási lépés

Az első finomítási lépésben a *termek közötti átjárást* fogjuk megadni:

- *P6: A termek elhelyezkedése meghatározza, hogy mely terem mely más termekkel és milyen irányban van kapcsolatban.*
  - ⋆ Ezek a kapcsolatok a modellben konstansként, egy bináris relációként jelennek meg:

$$com \in bld \leftrightarrow bld$$

- *P7: Egy terem saját magával nincs kapcsolatban.*

$$com \cap \mathrm{id}(bld) = \emptyset$$

- *P8: Egy személy csak akkor mehet át egy adott teremből (ahol tartózkodik) egy másikba, ha a két terem kapcsolatban van.*

A finomított esemény

A kiindulási modellben felvett pass esemény:

$$\begin{array}{rcl} & & \text{pass} \mathrel{\widehat{=}} \text{ANY } p, b \\ \text{WHERE } (p, b) & \in & aut \wedge sit(p) \neq b \\ \text{THEN } sit(p) & : & = b \text{ END} \end{array}$$

A finomítás:

$$\begin{array}{rcl} & & \text{pass} \mathrel{\widehat{=}} \text{ANY } p, b \\ \text{WHERE } (p, b) & \in & aut \wedge (sit(p), b) \in com \\ \text{THEN } sit(p) & : & = b \text{ END} \end{array}$$

A modellfinomítás konzisztenciája

A szükséges 4 feltétel:

1. Az invariánsok betartása: Ez a lépés itt nem ütközik nehézségekbe.

2. A konkrét esemény finomítása azt absztrakt eseménynek:

   - Az akció ugyanaz.
   - A konkrét feltételrész szigorúbb mint az absztrakt:

$$\begin{aligned} \exists(p,b) &: ((p,b) \in aut \wedge (sit(p),b) \in com) \Rightarrow \\ \exists(p',b') &: ((p',b') \in aut \wedge sit(p') \neq b') \end{aligned}$$

   mivel egy terem saját magával nincs kapcsolatban.

3. A konkrét események feltételrészeinek vizsgálata:

- A konkrét esemény nem fordul elő kevésbé gyakran, mint az absztrakt esemény: Az absztrakt esemény feltételrészéből következik a konkrét esemény feltételrésze:

$$\begin{aligned} \exists(p,b) &: ((p,b) \in aut \wedge sit(p) \neq b) \Rightarrow \\ \exists(p',b') &: ((p',b') \in aut \wedge (sit(p'),b') \in com) \end{aligned}$$

- Ez az állítás azonban nem bizonyítható:
  Ha tudjuk, hogy van egy személy, aki egy olyan terembe jogosult bemenni, ahol éppen nem tartózkodik, ebből nem következik, hogy lehet egy személy, aki olyan teremben tartózkodik, ami kapcsolatban van azzal, ahová jogosult bemenni.
- Ellenpélda:
  Egy személy egy adott teremből, ahol tartózkodik, nem tud kijutni, mert nincs jogosultsága egyik teremhez sem, amivel az adott terem kapcsolatban van (de van jogosultsága más távolabbi teremhez).
- A bizonyítás során felfedtünk tehát egy olyan esetet,
  ami a gyakorlati felhasználhatóság szempontjából is fontos.

4. Új események kizárólagosságának elkerülése:
   Mivel itt új eseményt nem vettünk fel,
   így ez a bizonyítási lépés nem szükséges.

További finomítás

A 3. feltétel megsértése miatt további megkötésekre van szükség.

- *P9: Egy személy nem maradhat bezárva egy teremben sem.*
  - ⋆ A formalizálás során kiindulhatunk a fent talált ellenpéldából, és invariánsként felírhatjuk:

$$aut \cap (sit; \overline{\mathrm{id}(bld)})) \neq \emptyset \Rightarrow (aut \cap (sit; com)) \neq \emptyset$$

  - ⋆ Ebből

$$\forall p : \exists b : ((p, b) \in aut \wedge (sit(p), b) \in com))$$

    Azaz minden esetben a terem, ahol egy adott személy tartózkodik, kapcsolatban van egy olyan teremmel, ahová a személy jogosult bemenni.

Hatások

- Ez a kitétel az esemény feltételrészét erősíthetné:
  - ⋆ Csak olyan terembe engedjük be a személyt, ahonnan ki is tud majd menni.
  - ⋆ Olyan feltételt kellene azonban keresni,
    amely a személy tartózkodási helyétől függetlenül megadható.
  - ⋆ Mivel már előírtuk a $sit \subseteq aut$ invariánst, erre lehetőségünk is van:
    $sit \subseteq (aut; com^{-1})$ bizonyításához
    elégséges az $aut \subseteq (aut; com^{-1})$ bizonyítása.
  - ⋆ Ez pedig a következőt jelenti:

$$\forall (p, b) : ((p, b) \in aut \Rightarrow \exists c : ((p, c) \in aut \wedge (b, c) \in com))$$

    tehát amennyiben egy p személy egy b teremben tartózkodik,
    ahol jogosultsága is van tartózkodni, akkor van olyan c terem,
    ahová jogosult bemenni, és c kapcsolatban van b-vel.
  - ⋆ Ez biztosítja, hogy p nem marad bezárva b-ben,
    hiszen c-n keresztül távozhat.

A tulajdonság felírása

Ez az állítás meg is szövegezhető:

- *P10: Minden személy, aki jogosult egy adott teremben tartózkodni, jogosult kell legyen egy olyan terembe is bemenni, amely terem kapcsolatban van az előzővel.*

A második finomítási lépés

A második finomítási lépésben bevezetjük a *forgóajtók kezelését*.

- *P11: A termek között egyirányú forgóajtók helyezkednek el. Minden ajtónak van egy bejárati és egy kijárati oldala.*
  - ⋆ Az ajtók: a dor halmaz (konstans).
  - ⋆ Az ajtó két oldalán lévő termek: az org és dst teljes függvények.
  - ⋆ Az ajtók a kapcsolatban lévő termek között vannak.

$$\begin{aligned} dor &\neq \emptyset \\ org &\in dor \to bld \\ dst &\in dor \to bld \\ com &= (org^{-1}; dst) \end{aligned}$$

- *P12: Egy forgóajtón akkor lehet átjutni, ha az nyitott.*
  A nyitás egyszerre csak egy személy részére történhet meg.

  ⋆ A *dap* részleges injektív függvény formalizálja
  (ajtó nyitható egy személy számára):

$$dap \in prs \rightarrowtail dor$$

- *P13: Egy forgóajtó nyitása egy adott személy részére*
  csak akkor történhet meg,
  ha a személy a bejárati oldalhoz tartozó teremben tartózkodik
  és jogosultsága van belépni a kijárati oldalon lévő terembe.

  ⋆ A beléptető rendszer fő funkcióját fogalmazza meg.
  ⋆ A következő predikátumok ezt az állítást formalizálják:

$$\begin{aligned} (dap; org) &\subseteq sit \\ (dap; dst) &\subseteq aut \end{aligned}$$

- *P14: Egy ajtóhoz tartozó zöld lámpa akkor gyullad ki, ha az ajtó nyitása megtörténik az adott személy részére.*

  ⋆ A dap függvény értékkészlete (vagyis $\mathrm{ran}(dap)$) azon ajtók halmaza, amelyek valamely személyek számára nyithatók.
  Ez éppen a kigyulladó zöld lámpák halmaza is lesz:

$$grn \widehat{=} \mathrm{ran}(dap)$$

- *P15: Egy ajtóhoz tartozó piros lámpa akkor gyullad ki, ha az ajtó nyitása nem történhet meg.*

- *P16: Egy adott ajtóhoz tartozó piros és zöld lámpa egyszerre nem gyulladhat ki.*

  ⋆ Így adódik a piros lámpák definíciója:

$$\begin{aligned} red &\subseteq dor \\ grn \cap red &= \emptyset \end{aligned}$$

- *P17: Egy személy egy ajtón átmehet,*
  ha az ajtó bejárati oldalán lévő teremben tartózkodik,
  és a kijárati oldalán lévő terembe jogosult belépni,
  valamint részére nem egy másik ajtó nyitása történik.

  ⋆ Ez az állítás az $admitted(p, q)$ predikátumot definiálja:

$$admitted(p, q) \mathrel{\widehat{=}} org(q) = sit(p) \wedge (p, dst(q)) \in aut \wedge p \notin \mathrm{dom}(dap)$$

Az események megfogalmazása: Áthaladás engedélyezése

- Az accept és refuse események
  a személyek áthaladásának engedélyezését fogalmazzák meg:

$$
\begin{array}{rcl}
 & & \text{accept} \mathrel{\widehat{=}} \text{ANY } p, q \\
\text{WHERE } p & \in & prs \wedge q \in dor \wedge q \notin grn \cup red \wedge admitted(p, q) \\
\text{THEN } dap(p) & : & = q \text{ END}
\end{array}
$$

$$
\begin{array}{rcl}
 & & \text{refuse} \mathrel{\widehat{=}} \text{ANY } p, q \\
\text{WHERE } p & \in & prs \wedge q \in dor \wedge q \notin grn \cup red \wedge \neg admitted(p, q) \\
\text{THEN } red & : & = red \cup \{q\} \text{ END}
\end{array}
$$

Az események megfogalmazása: Áthaladás

- Finomítjuk a pass eseményt,
  ami a személy mozgása mellett a zöld lámpa lekapcsolását is magába foglalja
  (a zöld lámpa a $dap()$ megváltozása miatt gyulladt ki az accept esemény hatására):

$$\begin{aligned} \text{pass} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in grn \\ \text{THEN } sit(dap^{-1}(q)) &:= dst(q) \parallel \\ dap &:= dap \triangleright \{q\} \text{ END} \end{aligned}$$

  Itt a $dap \triangleright \{q\}$ jelölés azt jelenti,
  hogy q-t kivesszük a $dap()$ értékkészletéből.

Az események megfogalmazása: Lámpa lekapcsolás

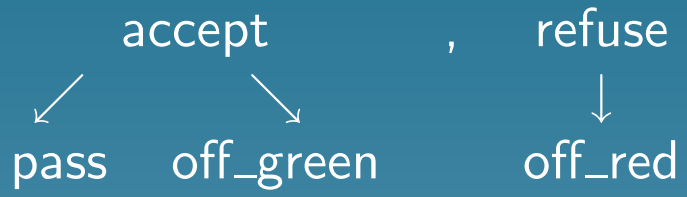
- Ezek után a zöld és a piros lámpa automatikus lekapcsolása marad hátra, ha senki sem halad át az ajtón:

$$\text{off\_grn} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q \in grn \text{ THEN } dap := dap \triangleright \{q\} \text{ END}$$

$$\text{off\_red} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q \in red \text{ THEN } red := red - \{q\} \text{ END}$$

Az események közötti szinkronizáció

- Az akciórészek és a feltételrészek által meghatározott szinkronizáció a következő:



Bizonyítás lépések

- A pass esemény finomítja az előző verziót.
- Az új események finomítják a skip eseményt.
- Ezek után két dolgot kell még megmutatni:
  - ⋆ (1) a pass esemény nem fordul elő kevésbé gyakran, mint az absztrakt pass esemény,
  - ⋆ (2) az új események nem "éheztetik ki" a pass eseményt.

Kiéheztetés

- A (2) bizonyítási kísérlete során új eshetőség kerül felszínre:
  A refuse és off_green események kiéheztethetik a pass esemény bekövetkezését!
  - ⋆ Feltételrészük folyamatosan igaz lehet ugyanakkor,
    mint a pass esemény feltételrésze.
- Példák:
  - ⋆ *Az a személy, aki az ajtó előtt áll és nincs jogosultsága átjutni,*
    újra és újra próbálkozik;
    ennek hatására olyan személy sem tud átjutni, akinek jogosultsága lenne.
  - ⋆ *Egy áthaladásra jogosult személy*
    az azonosítás és az ajtó nyitása után sem halad át a megadott ideig,
    hanem az ajtó automatikus záródása után folyamatosan
    újra próbálkozik és újra "lekési" az áthaladási időt.
- Ennek megoldása tervezői döntést igényel (pl. kártya elnyelése stb.)!

A harmadik finomítási lépés

A harmadik finomítási lépésben bevezetjük a *kártyaleolvasót*.

- Ennek feladata
  - ⋆ a kártyán lévő információ beolvasása (a személy azonosításához),
  - ⋆ ennek továbbítása a beléptetés vezérlőjéhez.
- A formalizálás során felmerülő, döntést igénylő kérdések:
  - ⋆ egyszerre hány kártya olvasható le,
  - ⋆ az áthaladásról való döntés közben betehető-e új kártya.

A kártyaleolvasó

D4: A kártyaleolvasó a kártya behelyezésétől az áthaladási (vagy áthaladás megtagadási) protokoll lezárásáig blokkolja a kártyabehelyező nyílást, így újabb kártya nem helyezhető be.

- A kártyaolvasó az ajtóval áll kapcsolatban,
  így nem is szükséges, hogy külön halmaz reprezentálja.
  A blokkolt leolvasókat a BLR jelöli:

$$BLR \subseteq dor$$

- A leolvasó és a beléptetés vezérlő közötti üzeneteket az $mCard$
  (részleges de nem injektív függvény)
  (kártya-adat küldése, a leolvasó/ajtó és a személy azonosítójával),
  valamint az $mAckn$ halmaz reprezentálja:

$$\begin{aligned} mCard &\in dor \rightsquigarrow prs \\ mAckn &\subseteq dor \end{aligned}$$

Az ajtó állapotai

Az olvasó blokkolt állapotában ajtó négy,
egymást kizáró állapot valamelyikében lehet:

- (1) az olvasó elküldte az $mCard$ üzenetet, de még nem jött válasz;
- (2) az ajtó nyitott, a zöld lámpa ég;
- (3) az ajtó zárt, a piros lámpa ég;
- (4) $mAckn$ üzenet érkezett, de az olvasó még nem dolgozta fel.

$$\begin{aligned} \mathrm{dom}(mcard) \cup grn \cup red \cup mAckn &= BLR \\ \mathrm{dom}(mCard) \cap (grn \cup red \cup mAckn) &= \emptyset \\ mAckn \cap (grn \cup red) &= \emptyset \end{aligned}$$

Az események felírása: Kártyaolvasás

- A kártyaolvasás mint "fizikai" esemény hatására blokkol az olvasó (a protokoll végéig) és elküldi az $mCard$ üzenetet:

$$\begin{aligned} \text{CARD} \mathrel{\widehat{=}} \text{ANY}\ \ p, q\ \text{WHERE}\ p &\in prs, q \in dor - BLR \\ \text{THEN}\ BLR &:= BLR \cup \{q\} \parallel \\ mCard &:= mCard \cup \{q \mapsto p\}\ \text{END} \end{aligned}$$

itt $q \in dor - BLR$ azt is jelöli,
hogy egy blokkolt olvasóba nem helyezhető újabb kártya.

Az események felírása: Áthaladás engedélyezése

- Az accept és refuse eseményeket finomítjuk az $mCard$ üzenetek kezelésével:

$$\begin{aligned} \text{accept} \mathrel{\widehat{=}} \text{ANY } p, q \text{ WHERE } (q, p) &\in mCard \wedge admitted(p, q) \\ \text{THEN } dap(p) &:= q \parallel \\ mCard &:= mCard - \{q \mapsto p\} \text{ END} \end{aligned}$$

$$\begin{aligned} \text{refuse} \mathrel{\widehat{=}} \text{ANY } p, q \text{ WHERE } (q, p) &\in mCard \wedge \neg admitted(p, q) \\ \text{THEN } red &:= red \cup \{q\} \parallel \\ mCard &:= mCard - \{q \mapsto p\} \text{ END} \end{aligned}$$

Az események felírása: Áthaladás

- A pass eseményt csak kis mértékben kell megváltoztatni:
  Az $mAckn$ üzenet elküldésére is szükség van:

$$\begin{aligned} \text{pass} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q \quad &\in \quad grn \\ \text{THEN } sit(dap^{-1}(q)) \quad &: \quad = dst(q) \parallel \\ dap \quad &: \quad = dap \rhd \{q\} \parallel \\ mAckn \quad &: \quad = mAckn \cup \{q\} \text{ END} \end{aligned}$$

Az események felírása: Áthaladás vége

- Az off_grn és off_red üzenetek ugyancsak elküldik az $mAckn$ üzenetet:

$$\begin{aligned} \text{off\_grn} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in grn \\ \text{THEN } dap &:= dap \rhd \{q\} \parallel \\ mAckn &:= mAckn \cup \{q\} \text{ END} \end{aligned}$$

$$\begin{aligned} \text{off\_red} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in red \\ \text{THEN } red &:= red - \{q\} \parallel \\ mAckn &:= mAckn \cup \{q\} \text{ END} \end{aligned}$$

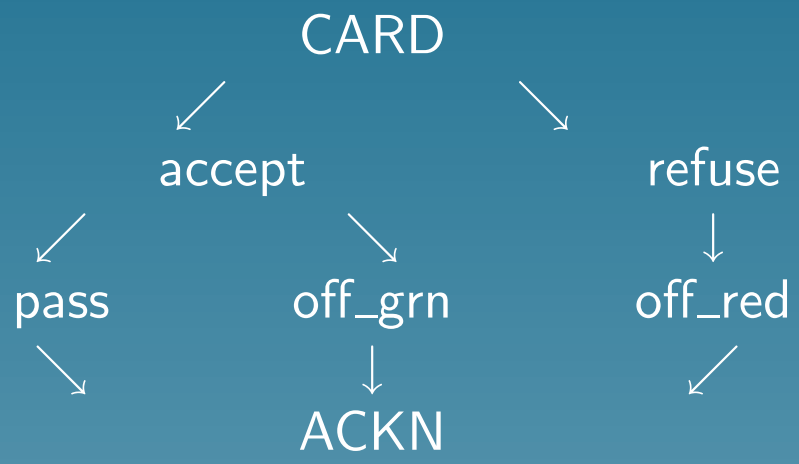
Az események felírása: Kártyaleolvasó blokkolása

- A kártyaleolvasó blokkolásának végét ugyancsak egy "fizikai" esemény reprezentálja, ami feldolgozza az $mAckn$ üzenetet is:

$$\begin{aligned} \text{ACKN} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mAckn \\ \text{THEN } BLR &:= BLR - \{q\} \parallel \\ mAckn &:= mAckn - \{q\} \text{ END} \end{aligned}$$

Az üzenetek szinkronizációja

- Az üzenetek szinkronizációja tehát a következő:



- A finomítási szabályok teljesítésének bizonyítása itt nem ütközik nehézségekbe.

A negyedik finomítási lépés

A *forgóajtókat* fogjuk modellezni:

- nyitás (elfogadás),
- zárás (elutasítás),
- áthaladás valamint a protokollt záró események.

A következő döntéseket hozhatjuk az ajtók fizikai kialakításával kapcsolatban:

- *D5: Az engedélyezési és áthaladási protokoll után az ajtó automatikusan bezáródik (blokkol).*
- *D6: Az ajtó rendelkezik lokális órával, ami adott nyitvatartás után lezárja az ajtót és lekapcsolja a zöld lámpát, illetve adott idő után lekapcsolja a piros lámpát.*

Új változók

- Jogosult személy esetén az ajtók nyitását vezérlő üzenetek: $mAccept$.
- A zöld lámpát kigyújtó nyitott ajtók: GRN.
- Az áthaladást jelző üzenetek: $mPass$.
- A nyitás utáni automatikus blokkolást jelző üzenetek: $mOff\_grn$ halmaz.

$$\begin{aligned} mAccept &\subseteq dor \\ GRN &\subseteq dor \\ mPass &\subseteq dor \\ mOff\_grn &\subseteq dor \end{aligned}$$

Az ajtó állapotai áthaladás közben

Lehetőségek:

- megkapta a nyitási üzenetet de még nem nyitott ki;
- kinyitott állapotban van és zöld lámpa világít;
- észlelte a személy áthaladását, lezárt és elküldte az erről szóló üzenetet;
- automatikusan lezárt és elküldte az erről szóló üzenetet:

$$
\begin{aligned}
mAccept \cup GRN \cup mPass \cup mOff\_grn &= grn \\
mAccept \cap (GRN \cup mPass \cup mOff\_grn) &= \emptyset \\
GRN \cap (mPass \cup mOff\_grn) &= \emptyset \\
mPass \cap mOff\_grn &= \emptyset
\end{aligned}
$$

Jogosulatlan személy azonosítása

Szükséges változók:

- Az áthaladás megtagadását jelző üzenetek: $mRefuse$.
- A zárt és piros lámpát kigyújtó ajtók:RED.
- A piros lámpa automatikus lekapcsolását jelző üzenetek: $mOff\_red$ halmaz.

$$\begin{aligned} mRefuse &\subseteq dor \\ RED &\subseteq dor \\ mOff\_red &\subseteq dor \end{aligned}$$

Az ajtó állapotai a tiltott ágon

Lehetőségek:

- megkapta az áthaladás elutasításáról szóló üzenetet,
  de még nem gyulladt ki a piros lámpa;
- kigyulladt a piros lámpa;
- lejárt az adott idő,
  lekapcsolt a piros lámpa és elküldte az erről szóló üzenetet:

$$\begin{aligned} mRefuse \cup RED \cup mOff\_red &= red \\ mRefuse \cap (RED \cup mOff\_red) &= \emptyset \\ RED \cap mOff\_red &= \emptyset \end{aligned}$$

Az események felírása: Nyitás

- A vezérlő accept eseménye küldi el a nyitást jelző $mAccept$ üzenetet (a jogosult személy azonosítása után, ami a kártyaolvasó $mCard$ üzenete alapján történt):

$$\begin{aligned} \text{accept} \mathrel{\widehat{=}} \text{ANY } p, q \text{ WHERE } (q,p) &\in mCard \wedge admitted(p,q) \\ \text{THEN } dap(p) &: = q \parallel \\ mCard &: = mCard - \{q \mapsto p\} \parallel \\ mAccept &: = mAccept \cup \{q\} \text{ END} \end{aligned}$$

- Ennek hatására az ajtó ACCEPT eseménye valósítja meg a nyitást:

$$\begin{aligned} \text{ACCEPT} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mAccept \\ \text{THEN } GRN &: = GRN \cup \{q\} \parallel \\ mAccept &: = mAccept - \{q\} \text{ END} \end{aligned}$$

Az események felírása: Áthaladás

- Az áthaladás fizikai megvalósulása:

$$\begin{aligned} \text{PASS} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in GRN \\ \text{THEN } GRN &:= GRN - \{q\} \parallel \\ mPass &:= mPass \cup \{q\} \text{ END} \end{aligned}$$

- Az ezt regisztráló működést a vezérlőben
(pass esemény az $mAckn$ nyugtaküldéssel a kártyaolvasó nyitásához):

$$\begin{aligned} \text{pass} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mPass \\ \text{THEN } sit(dap^{-1}(q)) &:= dst(q) \parallel \\ dap &:= dap \rhd \{q\} \parallel \\ mAckn &:= mAckn \cup \{q\} \parallel \\ mPass &:= mPass - \{q\} \text{ END} \end{aligned}$$

Az események felírása: Záródás

- Ha nem volt áthaladás, akkor az automatikus záródás következik be. Ezt a vezérlőnek az $mOff\_grn$ üzenet jelzi.

$$\begin{aligned} \text{OFF\_GRN} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in GRN \\ \text{THEN } GRN &:= GRN - \{q\} \parallel \\ mOff\_grn &:= mOff\_grn \cup \{q\} \text{ END} \end{aligned}$$

- Ezt regisztrálja a vezérlő (off_grn) és nyugtát küld az $mAckn$ üzenettel:

$$\begin{aligned} \text{off\_grn} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mOff\_grn \\ \text{THEN } dap &:= dap \rhd \{q\} \parallel \\ mAckn &:= mAckn \cup \{q\} \parallel \\ mOff\_grn &:= mOff\_grn - \{q\} \text{ END} \end{aligned}$$

Az események felírása: Áthaladás megtagadása

- Ha a személy az áthaladásra nem jogosult
  (ami a kártyaolvasó $mCard$ üzenete alapján derül ki),
  akkor a vezérlő refuse eseménye küldi el az ajtónak az $mRefuse$ üzenetet:

$$\begin{aligned} \text{refuse} \mathrel{\widehat{=}} \text{ANY } p, q \text{ WHERE } (q, p) &\in mCard \wedge \neg admitted(p, q) \\ \text{THEN } red &:= red \cup \{q\} \parallel \\ mCard &:= mCard - \{q \mapsto p\} \parallel \\ mRefuse &:= mRefuse \cup \{q\} \text{ END} \end{aligned}$$

- Ennek hatására az ajtó zárva marad és kigyújtja a piros lámpát:

$$\begin{aligned} \text{REFUSE} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mRefuse \\ \text{THEN } RED &:= RED \cup \{q\} \parallel \\ mRefuse &:= mRefuse - \{q\} \text{ END} \end{aligned}$$

Az események felírása: Piros lámpa kikapcsolás

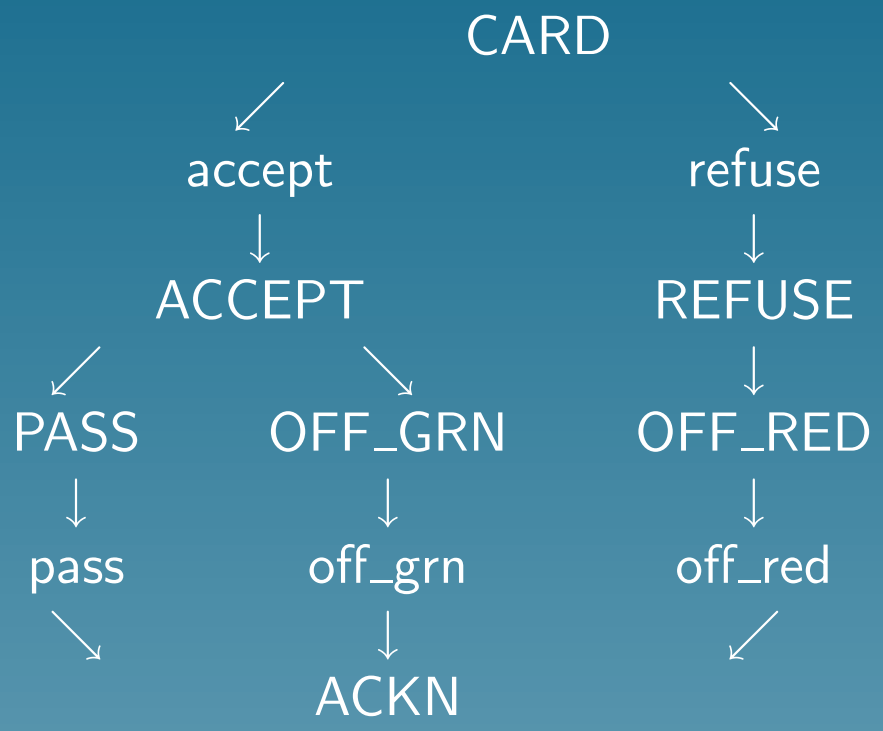
- A piros lámpa kikapcsolása és az erről szóló $mOff\_red$ üzenet elküldése:

$$\begin{aligned} \text{OFF\_RED} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in RED \\ \text{THEN } RED &:= RED - \{q\} \parallel \\ mOff\_red &:= mOff\_red \cup \{q\} \text{ END} \end{aligned}$$

- A vezérlő regisztrál (off_red esemény)
  és $mAckn$ nyugtát küld a kártyaolvasó nyitásához:

$$\begin{aligned} \text{off\_red} \mathrel{\widehat{=}} \text{ANY } q \text{ WHERE } q &\in mOff\_red \\ \text{THEN } red &:= red - \{q\} \parallel \\ mAckn &:= mAckn \cup \{q\} \parallel \\ mOff\_red &:= mOff\_red - \{q\} \text{ END} \end{aligned}$$

Az események szinkronizálása



- Az ábrán jól látható a fizikai események (nagybetűs nevek) és a vezérlő viselkedésének az interakciója.
- A finomítási szabályok bizonyítása itt sem ütközik nehézségekbe.

Szekvenciális programok konstrukciója

Cél:

- Szekvenciális programok automatikus konstrukciója.
- Konstrukciós szabályok + az előrevetítés (anticipation) technikája.

Szekvenciális programok:

- Állítások: műveletek, adatkezelés, számítások
- Vezérlési szerkezetek: sorrendezés, elágazások, iterációk.

Szekvenciális programok hagyományos fejlesztése

- Kezdeti algoritmus-váz + fokozatosan finomítás a futtatható kódig.
- A finomítás minden lépése ütemezett utasításokból áll.
- A saját absztrakciós szintjén minden lépés teljesnek mondható (bár sokszor még nem-determinisztikus).

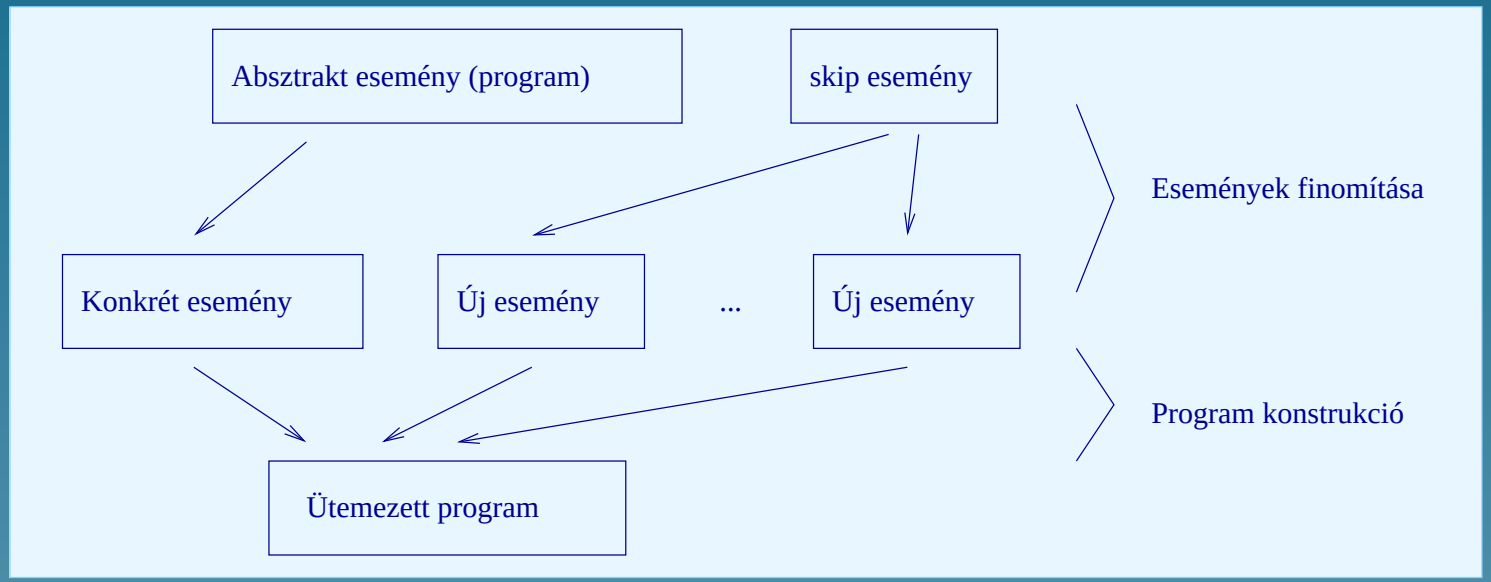
A most bemutatandó megközelítés

- A matematikai logikai alapokon való fejlesztés lényege:
  az utasítások kidolgozásának és a vezérlésnek a szétválasztása.

- Először: Az utasítások felvétele és finomítása.
  - ⋆ Állítások = események: feltételrész és akciórész.
  - ⋆ Párhuzamos, elosztott végrehajtás (nincs ütemezés):
    feltételrész határozza meg a végrehajthatóságot.
  - ⋆ Finomítás: feltételrész szigorúbbá tétele, új esemény felvétele.
  - ⋆ A program egyes részeit függetlenül lehet kidolgozni.
  - ⋆ Bizonyítható a finomítás konzisztenciája.

- Ha ez kész: Ütemezés, vagyis a vezérlési szerkezetek konstrukciója
  - ⋆ Szisztematikusan, konstrukciós szabályok alapján.
  - ⋆ Az egyes események összeillesztése a feltételrészek alapján
    (így a feltételrészek egy része gyakorlatilag el is tűnik).

Program konstrukció eseménybizonyítással



A program struktúrája

Kezdetben ismert váétozók és konstansok:

- Bemeneti paraméterek (típussal) és előfeltétel:

$$parameters \in S_p \wedge Pre\_condition$$

- Várható kimenetek (típussal) és az utófeltétel:

$$results \in S_r \wedge Post\_condition$$

Kezdeti események

- A kezdeti modell egy eseményt tartalmaz:

$$\text{aprog} \mathrel{\widehat{=}} \text{BEGIN } results : (results \in S_r \wedge Post\_condition) \text{ END}$$

- Inicializáló esemény, ami a kimeneteket "szabadon választhatóvá" teszi:

$$\text{init} \mathrel{\widehat{=}} \text{BEGIN } results : (results \in S_r) \text{ END}$$

Fejlesztési folyamat

- A feltételrészek szigorúbbá tétele + új események felvétele.
- A finomítás konzisztenciáját szolgáló szabályok betartása.
- Élő és terminálló programok létrehozásához:
  - ⋆ A feltételrészek diszjunkciója igaz (az adott invariánsok mellett).
  - ⋆ Az újonnan bevezetett események nem gátolják meg állandó jelleggel más események végrehajtását.
  - ⋆ Az akciórészek egy jól megalapozott halmaz elemét csökkentik (olyan, részben rendezett halmaz, amelyben nem létezik végtelen, csökkenő elemekből álló szekvencia; lásd pl. a természetes számok).

A fejlesztési folyamat vége

- A finomítással előállított események összerakása konstrukciós szabályok segítségével (egy esemény lesz):

$$\mathrm{cprog} \mathrel{\widehat{=}} \mathrm{BEGIN}\ \mathit{Initialization};\ \mathit{Program}\ \mathrm{END}$$

  Itt:

- Initialization az init esemény,
- Program az aprog esemény alapján.

A finomítás konstrukciós szabálya

- skip eseményt finomító új U esemény bevezetése.
- Egy természetes szám csökkentése:
  U nem gátol folyamatosan más eseményeket.

$$
\begin{array}{rcl}
S & \rightsquigarrow & S' \sqcup U, \text{ ahol} \\
S & \sqsubseteq & S' \\
skip & \sqsubseteq & U \\
I & \Rightarrow & \mathrm{guard}(S') \vee \mathrm{guard}(U) \\
I & \Rightarrow & V \in N \\
I & \Rightarrow & (V = n \Rightarrow [U](V < n))
\end{array}
$$

- Itt $\sqcup$ jelöli a választást (események implicit ütemezése),
  $S \sqsubseteq S'$ jelöli azt, hogy S' finomítása S-nek.

A konstrukciós szabályok

- Egy vagy több eseményből a program vezérlési szerkezetét állítják össze.
- Itt a strukturált programokban megszokott szerkezeteket használjuk:
  - ⋆ IF Q THEN S ELSE T END
  - ⋆ WHILE Q DO S END
- Jelölés: $P \Longrightarrow S$ rövidíti a SELECT P THEN S END eseményt.

A feltételes elágazás szabálya

Két eseményből egy IF ... THEN ... ELSE ... END elágazást állít össze:

$$\begin{aligned}(P \wedge Q &\implies S) \sqcup (P \wedge \neg Q \implies T) \sqcup U \rightsquigarrow \\ (P &\implies \text{IF } Q \text{ THEN } S \text{ ELSE } T \text{ END}) \sqcup U\end{aligned}$$

Az iteráció szabálya

Eseményeket alakít át WHILE ... DO ... END iterációvá:

$$\begin{aligned} (P \wedge Q &\implies S) \sqcup (P \wedge \neg Q \implies T) \sqcup U \rightsquigarrow \\ (P &\implies \text{WHILE } Q \text{ DO } S \text{ END}; T) \sqcup U \\ \text{ahol } I \wedge P \wedge Q &\Rightarrow [S]P \text{ valamint } S \text{ és } T \text{ esetén nincs feltételrész} \end{aligned}$$

Az utolsó feltétel: Az iteráció során P invariáns.

Egyszerűsített forma:

$$\begin{aligned} (Q &\implies S) \sqcup (\neg Q \implies \text{skip}) \sqcup U \rightsquigarrow \\ &(\text{WHILE } Q \text{ DO } S \text{ END}) \sqcup U \\ &\text{ahol } S \text{ esetén nincs feltételrész.} \end{aligned}$$

Új esemény bevezetése

- Finomítás: új U esemény (skip finomítása) bevezetése.
- Természetes szám változó csökkentése
  (vagy más, jól megalapozott halmazból választott változó).
- Ez a változó a finomítás során újonnan bevezetett is lehet.

Tegyük fel: a finomítás $i+1$-edik lépésében vezetjük be

- az Eventx eseményt,
- az S_y halmazon értelmezett y változót
  (az ehhez tartozó, állapotra vonatkozó finomítási invariánssal együtt),
- y-t az Eventx esetén az említett csökkentés előírására fogjuk felhasználni,
  mégpedig a $V(y)$ kifejezésben.

Az előrelátás technikája

- Már az *előző finomítási lépésben* (i-edik lépésben) bevezetjük az y változót és az Eventx eseményt a következő formában:

$$\text{Eventx} \mathrel{\widehat{=}} \text{BEGIN } y : (y \in S_y \wedge V(y) < V(y_0)) \text{ END}$$

  Ez teljesíti a finomítási szabályokat (skip finomítása).

- Az $i+1$-edik lépésben, "normál" finomításként kerül sorra Eventx konkrét formájának megfogalmazása.

- Ekkor bizonyítjuk be a finomítás konzisztenciáját.

- Egyszerűsödnek a bizonyítási lépések.

Mintapélda: Adatkeresés

Egy indexekkel ellátott tömbben keressük egy megadott tömbelem indexét.

Változók és konstansok:

- S a tömbelemek halmaza,
- n (tömbelemek száma), f (a tömb) és x (a megadott tömbelem) három konstans,
- i (a keresett index) egy változó:

$$\begin{aligned} n &\in N_1 \\ f &\in 1..n \to S \\ x &\in \mathrm{ran}(f) \\ i &\in 1..n \end{aligned}$$

Kiindulás

- A kezdeti esemény:

$$\text{aprog} \mathrel{\widehat{=}} \text{BEGIN } i : (i \in 1..n \wedge f(i) = x) \text{ END}$$

- Az előre látott esemény:

$$\text{progress} \mathrel{\widehat{=}} \text{BEGIN } \ i : (i \in 1..n \wedge n - i < n - i_0) \text{ END}$$

- Nem használunk külön változót a finomított eseményben, magát az i változót növeljük majd (így $n - i$ értékét csökkentve).

- Megközelítés: Az indexeket növelve haladunk a tömbben, ha az $i - 1$ indexig nem találtuk meg a keresett elemet, akkor lépünk tovább az i-edik elemre.

A finomítás eredménye

- A finomított események:

$$\text{init} \mathrel{\widehat{=}} \text{BEGIN } i := 1 \text{ END}$$

$$\text{aprog} \mathrel{\widehat{=}} \text{SELECT } f(i) = x \text{ THEN skip END}$$
$$\text{progress} \mathrel{\widehat{=}} \text{SELECT } \ f(i) \neq x \text{ THEN } i := i + 1 \text{ END}$$

- A finomítás konzisztenciája bizonyítható.

- Az aprog esemény nem "keres" többé,
  hanem tulajdonképpen a megállást írja le.

A program konstrukciója

- Az iteráció szabálya: az aprog és progress eseményekből egy iteráció áll elő.
- Ehhez fűzhető az inicializálást végző esemény.

$$\mathrm{cprog} \mathrel{\widehat{=}} \mathrm{BEGIN}\ i := 1; \mathrm{WHILE}\ f(i) \neq x\ \mathrm{DO}\ \ i := i + 1\ \mathrm{END}\ \ \mathrm{END}$$

Mintapélda: Maximumkeresés

Egy természetes számokat tartalmazó tömbben keressük meg a maximális értéket!

Változók és konstansok:

- n (tömbelemek száma) és f (maga a tömb) egy-egy konstans,
- m (a maximum érték) egy változó,
- míg k (a maximum érték indexe) egy előrevetített változó:

$$\begin{aligned} n &\in N_1 \\ f &\in 1..n \to N \\ m &\in \mathrm{ran}(f) \\ k &\in 1..n \end{aligned}$$

Kiindulás

- A kezdeti esemény:

$$\text{aprog} \mathrel{\widehat{=}} \text{BEGIN } m : (m \in \text{ran}(f) \wedge \forall x : (x \in 1..n \Rightarrow f(x) \leq m)) \text{ END}$$

- Két előrevetített esemény:

$$\text{test1} \mathrel{\widehat{=}} \text{BEGIN } m, k : (m \in \text{ran}(f) \wedge k \in 1..n \wedge n - k < n - k_0) \text{ END}$$

$$\text{test2} \mathrel{\widehat{=}} \text{BEGIN } m, k : (m \in \text{ran}(f) \wedge k \in 1..n \wedge n - k < n - k_0) \text{ END}$$

  Mindkét esemény csökkenti $n - k$ értékét;
  m változó pedig szabadon választható az értelmezési tartományából.

- Kimerítő keresés: A következő invariánst írjuk fel:

$$\forall x : (x \in 1..k \Rightarrow f(x) \leq m)$$

A finomítás

A finomítás során a következő eseményeket írjuk fel:

$$\text{init} \mathrel{\widehat{=}} \text{BEGIN } k := 1 \parallel m := f(1) \text{ END}$$

$$\text{aprog} \mathrel{\widehat{=}} \text{SELECT } k = n \text{ THEN skip END}$$

$$\text{test1} \mathrel{\widehat{=}} \text{SELECT } k \neq n \wedge f(k+1) \leq m \text{ THEN } k := k+1 \text{ END}$$

$$\text{test2} \mathrel{\widehat{=}} \text{SELECT } k \neq n \wedge f(k+1) > m \text{ THEN } k := k+1 \parallel m := f(k+1) \text{ END}$$

A finomítás konzisztenciája itt is bizonyítható.

A program vezérlési struktúra

- A test1 és test2 eseményekből a feltételes elágazás szabálya,
- az aprog eseménnyel az iterációs szabály,
- majd az init esemény:

cprog $\widehat{=}$ BEGIN $k, m := 1, f(1)$

WHILE $k \neq n$ DO

IF $f(k+1) \leq m$ THEN $k := k+1$ ELSE $k, m := k+1, f(k+1)$ END

END

END

A B-módszer alapjai

- A B-módszer kidolgozása: J.-R. Abrial, 1985-1992, a Z továbbfejlesztése.
- Alapok: E.W. Dijkstra és C.A.R Hoare munkái.
- A teljes elméleti alapozás és a módszer leírása: a *B book*.
- A matematikai modellen alapú szoftverfejlesztési módszerek közé tartozik:
  - ⋆ a VDM módszerhez hasonló,
  - ⋆ könnyebben használható (?),
  - ⋆ hatékony eszköztámogatás.

A B módszer

- *Jelölésrendszert* és *módszert* ad rendszerek specifikációjához és tervezéséhez.
- A tervezés a kiindulási specifikáció lépésenkénti finomításával történik.
- Az egyes finomítási lépések, illetve az eredményként előálló specifikáció ellenőrzéséhez *matematikai apparátus* áll rendelkezésre (kritériumok és bizonyítási szabályok készlete).
- A használt formalizmus az úgynevezett absztrakt állapotgép (*abstract state machine*).
  - ⋆ Ennek alapja Dijkstra feltételekkel védett parancsnyelve (*guarded command language*).
  - ⋆ Az egyes utasításokhoz elő- és utófeltételek adhatók meg.
  - ⋆ Az adatleíró nyelv pedig a halmazelmélet jól ismert fogalmaira épül.

Hibamentes szoftvertervezés (*zero defect design*)

A B-módszer és a tipikus szoftverfejlesztési folyamat kapcsolata:

- Követelmény-specifikáció:
  A B-módszer a követelmények formalizálásában
  és ezek konzisztenciájának megállapításában segít.

- Architektúratervezés és részletes tervezés:
  A B-módszer a modellfinomítási lépések helyességének vizsgálatával
  egészíti ki ezt a folyamatot.

  ⋆ A kezdeti modell finomítása történik,
  míg el nem jutunk a felhasznált programozási nyelvi konstrukciók
  illetve könyvtári elemek szintjére,
  amikor a közvetlen megvalósítás (kódgenerálás) lehetséges.

Az absztrakt állapotgép

Az absztrakt állapotgép elemei:

- a globális kényszerek (*constraints*),
- az előre ismert konstansok (*constants*),
- a tulajdonságok (*properties*),
- az állapot megadására használt változók (*variables*),
- az ezekre vonatkozó invariánsok (*invariants*),
- az inicializálás (*initialization*),
- az események/műveletek leírása (*operations*),
  itt az előfeltételek (*preconditions*) és az eredmények.

Formalizálás

MACHINE $Machine\_name(p)$

CONSTRAINTS $Cst(p)$

CONSTANTS c

PROPERTIES $Ctx(c, p)$

VARIABLES v

INVARIANT $Inv(p, c, v)$

INITIALISATION $Init$

OPERATIONS $Operation\_name$ PRE $Pre(p, c, v)$ THEN St END

END

- Cst, Ctx, Inv és Pre elsőrendű logikai predikátumok,
- p, c, v változók listája, St pedig egy művelet.

A specifikáció konzisztenciájának bizonyítása

- A specifikált környezet (konstansok, formális paraméterek) létezik:

$$\begin{aligned} \exists p &: Cst(p) \\ Cst(p) &\Rightarrow \exists c : Ctx(c,p) \\ Cst(p) \wedge Ctx(p,c) &\Rightarrow \exists v : Inv(p,c,v) \end{aligned}$$

- Az inicializálás biztosítja az invariánsok teljesülését:

$$Cst(p) \wedge Ctx(p,c) \Rightarrow [Init]Inv(p,c,v)$$

- Minden esemény (művelet) megtartja ezeket az invariánsokat:

$$Cst(p) \wedge Ctx(p,c) \wedge Inv(p,c,v) \wedge Pre(p,c,v) \Rightarrow [St]Inv(p,c,v)$$

$[St]R$ jelentése: St végrehajtása teljesíti az R predikátumot.

Műveletek és predikátumok

Az St műveletek és az ekvivalens predikátumok:

Művelet megadása B-ben	Ekvivalens predikátum
[BEGIN S END]R	$[S]R$
[PRE P THEN S END]R	$P \wedge [S]R$
[CHOICE S OR ... OR T END]R	$[S]R \wedge ... \wedge [T]R$
[IF P THEN S ELSE T END]R	$(P \Rightarrow [S]R) \wedge (\neg P \Rightarrow [T]R)$
[IF P THEN S END]R	$(P \Rightarrow [S]R) \wedge (\neg P \Rightarrow R)$
[ANY l WHERE P THEN S END]R	$\forall l : (P \Rightarrow [S]R)$, ha l kötött R-ben
[VAR l IN S END]R	$\forall l : [S]R$, ha l kötött R-ben
$[v := e]R$	R, ahol v helyett e szerepel

- Minden St művelet mint egy predikátum-transzformátor értelmezhető, amely új predikátumot állít elő.

- Ezeket a táblázatban felírt szabályokat helyettesítési szabályoknak is felfoghatjuk, amik axiomatizálhatók.

Bizonyítandó állítások

- A pszeudo-nyelvű leírásokból
  elsőrendű logikai illetve halmazelméleti állítások képezhetők.

- Ezeket a bizonyítandó állításokat a B eszközkészlet automatikusan generálja.

A formalizmus néhány tulajdonságát külön is kiemeljük:

- Az absztrakt állapotgépek paraméterezhetők (lásd p),
  ugyanaz a specifikáció más paraméterrel újra használható.

- A CHOICE helyettesítés a nemdeterminisztikus modellezés lehetőségét adja.

- Az ANY lehetővé teszi általános feltételek használatát.

- A választás $\|$ operátora segítségével kombinálhatók műveletek,
  így összetett helyettesítések adhatók meg.

A specifikáció strukturálása

- Egy absztrakt állapotgép beágyazható egy másik állapotgépbe,
  azzal a kikötéssel, hogy a beágyazott állapotgép változói
  nem módosíthatók a beágyazó állapotgépből (invariánsokban hivatkozhatók).

- A beágyazott változókat csak a beágyazott állapotgép műveletei módosíthatják.

- Így a beágyazott állapotgépre vonatkozó invariánsok nem adódnak hozzá
  a beágyazó állapotgépre vonatkozó *bizonyítandó* állításokhoz.

- A matematikai alap:

$$[Op1_{m1}]Inv_{m1} \wedge [Op2_{m2}]Inv_{m2} \Rightarrow [Op1_{m1} \parallel Op2_{m2}](Inv_{m1} \wedge Inv_{m2})$$

  ahol $Op1_{m1}$ az $m1$ állapotgép egy $Op1$ műveletét jelenti, $Op2_{m2}$ hasonlóképpen.
  Az összetett specifikáció "örökli" a komponensek (bizonyított) tulajdonságait.

További szintaktikai elemek a strukturáláshoz

A beágyazott állapotgépek paraméterezhetősége és átnevezhetősége:

- PROMOTES kulcsszó: a beágyazott állapotgép azon műveletei, amelyek (változás nélkül) az összetett állapotgép műveletei lesznek.
- Beágyazás: INCLUDE
- Láthatóság: SEES
  (a csak látható változók invariánsokban sem használhatók fel)
- Használhatóság: USES relációk
  (invariánsokban használható, de nem módosítható a használó állapotgépben).

A B eszközkészlet

- B-Tool (1991, BP), *Atelier B Tools* (Steria Mediterranee, 1994), *B Toolkit* (B-Core Ltd.)
  - ⋆ Szintaxis- és típusellenőrzés (Analyser, TypeChecker).
  - ⋆ Automatikus állításgeneráló (Proof Obligaton Generator)
  - ⋆ Háromféle bizonyító rendszer:
    - ∗ automatikus (Autoprover),
    - ∗ interaktív (InterProver)
    - ∗ felhasználói (BToolProver)
  - ⋆ A specifikáció futtatása és tesztelése modell animátor segítségével (Animator).
  - ⋆ C programkód generálása (Translator).
    - ∗ Könyvtári elemek is részt vesznek, pl. a ki- bemeneti kódrészletek (InterfaceGenerator)
    - ∗ A kódrészleteket összekapcsolása és fordítása (Linker).
  - ⋆ LaTeX dokumentáció generálás (DocumentMarkUp).
  - ⋆ Verziókezelés és a függőségek nyilvántartása (Manager).

Vizuális modellezés és a B kapcsolata

- A B-módszer integrálható az UML alapú szoftverfejlesztési folyamattal: verifikációs lépéseket tudunk a B segítségével elvégezni.

- A B által használt absztrakt állapotgép formalizmus: UML diagramokból transzformációval előállítható.

Strukturális diagramok

Az osztálydiagramok használhatók fel:

- A B állapotgépeket az UML osztályai azonosítják.
- Az attribútumok kezdeti értéke: a B specifikáció inicializáló részében (INITIALISATION).
- OCL kifejezések és a társításokhoz rendelt számosság: invariánsok (INVARIANTS).
- A metódusokhoz rendelt elő-és utófeltételek: B műveletekben (OPERATIONS).
- Állapotgépek összekapcsolása:
  az osztályok közötti kapcsolatok csak fa struktúrájúak lehetnek.

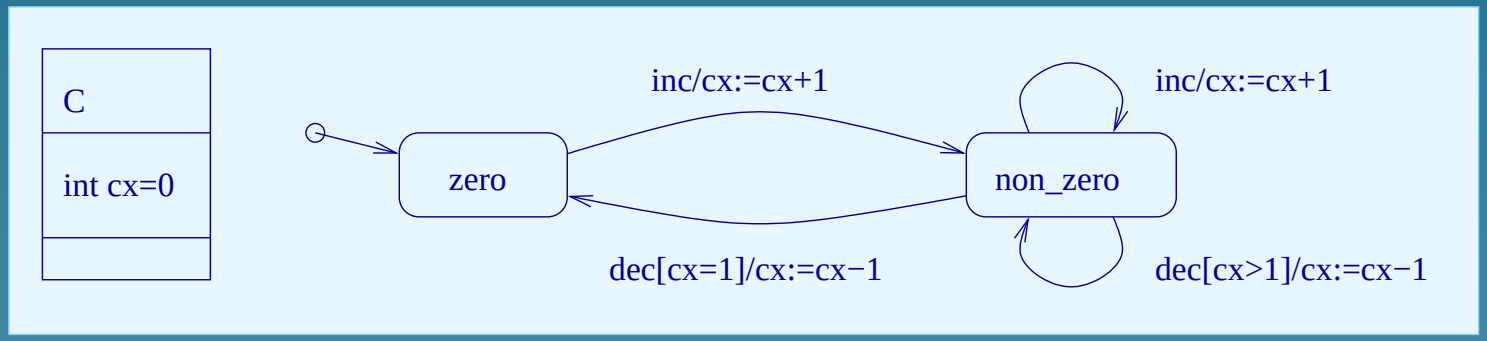
Viselkedési diagramok

Az osztályokhoz rendelt viselkedési diagramok: műveletek (OPERATIONS).

- Az állapottérkép állapotai: felsorolás típusú állapotváltozó.
- Események: szintén állapotváltozók.
- Állapotátmenetek: egy-egy műveletet azonosítanak
  - ⋆ az állapotváltozókon értelmezettek,
  - ⋆ az átmenet kiindulási állapota: előfeltétel,
  - ⋆ trigger és őrfeltétel: ugyancsak előfeltétel,
  - ⋆ akció és célállapot: értékadás.

Példa UML diagram

Példaként tekintsük a C nevű osztályt, aminek állapottérkép diagramja az alábbi:



A B specifikáció elemei

A B specifikáció állapotváltozói:

- A C osztály példányainak azonosítói ($Cinstances$).
  - ⋆ A lehetséges példányokat egy halmaz reprezentálja ($CSET$)
  - ⋆ Egy példány kijelölése ($thisC$ "index").
- Az állapot-azonosító (C_state)
  - ⋆ Az állapotok halmaza (C_STATE).
- Az osztály attribútuma (cx).
  - ⋆ Típus hozzárendelése itt is.

Az invariánsok megkötik az állapotváltozókat.

A B specifikáció

MACHINE C
SETS $CSET; C\_STATE = \{zero, non\_zero\}$
VARIABLES $Cinstances, c\_state, cx$
INVARIANT
$Cinstances \subseteq CSET$ &
$c\_state : Cinstances \rightarrow C\_STATE$ &
$cx : Cinstances \rightarrow NAT$
INITIALISATION $Cinstances := \{\} \parallel c\_state := \{\} \parallel cx := 0$

Műveletek

A dec művelet:

$dec(thisC) =$
PRE $thisC : Cinstances$ THEN
SELECT $c\_state(thisC) = non\_zero \,\&\, cx(thisC) = 1$
THEN $c\_state(thisC) := zero$ END
|| SELECT $c\_state(thisC) = non\_zero \,\&\, cx(thisC) > 1$
THEN $skip$ END
|| $cx(thisC) := cx(thisC) - 1$ END
END

- Az UML profile használatával az UML diagramjaiból B specifikáció generálható.
- Így az UML modell konzisztenciája és a finomítási lépések helyessége a B eszközök segítségével vizsgálható.

Felhasználási példák

- GEC Alstom, Matra Transport: vasúti és metró fejlesztések.
- TA Group Ltd.: ejtőernyő aktiválási rendszer.
- IBM: CISC/ESA rendszer, szoftver modellezési eszköz.
- Atomic Weapon Establishment: fegyverzet kezelő szoftver fejlesztése.
- SAET Meteor: biztonsági rendszer modellezése:
  - ⋆ több tízezer soros B kód,
  - ⋆ az Atelier B Tools állításgenerátora: 30.000 állítás,
  - ⋆ a bizonyítások kb. 80%-a automatikusan elvégezhető.