5. gyakorlat – Modellek ellenőrzése és tesztelése – Megoldások

1. feladat

Az `f()` függvénnyel szemben a következő követelményeink vannak:

- R1. Az `f()` függvénynek minden végrehajtása során legalább egyszer outputot kell kiadnia.
- R2. Az `f()` függvénynek tetszőleges inputsorozat esetén terminálnia kell.
- R3. Az `f()` függvény végrehajtása során kiadott legutolsó output értéke kötelezően 0.

A függvény egy lehetséges megvalósítását adja meg az alábbi C nyelvű kódrészlet:

```
int readInput();
void writeOutput(int out);

void f() {
  int x = readInput();
  int y = readInput();
  int z = x + y;
  writeOutput(x * y);
  while (x > 0 && y > 0) {
    if (1 == readInput() % 2) {
      y--;
      z--;
    } else {
      x--;
      y++;
    }
    writeOutput(z + x * y * y - x - y);
  }
}
```

A következő lépések során ellenőrizzük a függvény működését!

a. Ábrázoljuk folyamatmodellként `f()` vezérlési folyamát!
b. Miért lehetünk biztosak az R1 teljesülésében?
c. Miért lehetünk biztosak az R2 teljesülésében?
d. Építsünk olyan állapotgépet, amely az `f()` függvénnyel ekvivalens módon működik. Modellezzük a `readInput()` hívásokat input csatornaként, valamint a `writeOutput()` hívást output csatornaként. Az `f()` függvény terminálását modellezzük úgy, hogy az automata ad egy speciális outputot, és átmegy egy nyelő (kimenő átmenet nélküli) állapotba.
e. Az R3 követelményt teszteléssel ellenőrizzük. A $t_1 = \langle 2, 3, 5, 7, 11, 13, ... \rangle$ input szekvencia a tesztesetünk. Detektálunk-e hibát?
f. Számítsunk *utasításszintű tesztfedést*, vagyis hogy az utasítások mekkora hányadát járja be a tesztelt függvény a teszteset végrehajtása során! Hogy jelenik meg ez a mérőszám a vezérlési folyamon?
g. Az R3 követelményt teszteléssel ellenőrizzük. A $t_2 = \langle 1, 2, 4, 1, 2, 4, ... \rangle$ input szekvencia a tesztesetük. Detektál-e hibát ez a teszteset? Mekkora a két tesztből álló tesztkészlet együttes utasításfedése?
h. Milyen tesztfedettségi metrika számítható a korábban megépített állapotgép alapján?
i. Készítsünk olyan *tesztorákulum* automatát, amely `f()` input és output szekvenciái és terminálása alapján el tudja dönteni, hogy az adott lefutás során az R3 követelmény sérült-e! Hogy viselkedik az orákulum a fenti tesztinputra?
j. Kimaradt-e a fedésből a vezérlési folyam, ill. az automatamodell bármelyik része? Milyen fedési metrika mutathatja ezt ki?

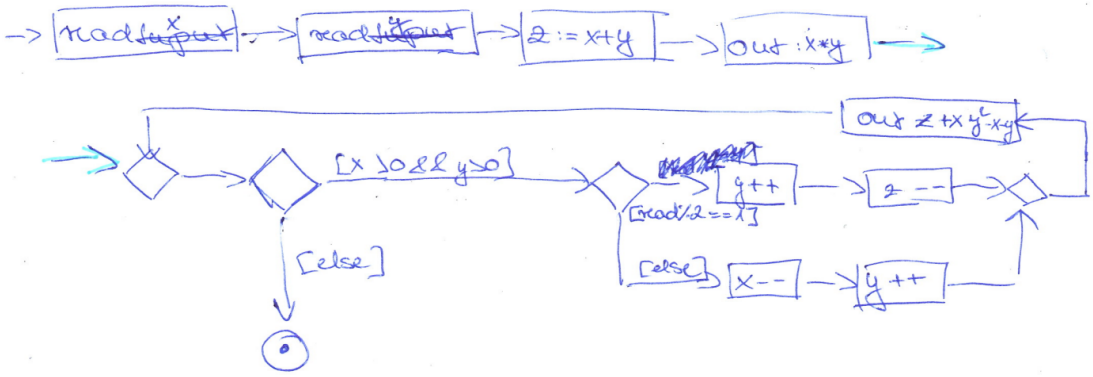
k. Adjunk meg egy tesztesetet, amely kimutat egy hibát a programban! Milyen elv alapján sejthettük volna meg, hogy a korábban összeállított tesztkészletünk kiegészítésre szorul?

l. *Otthoni munka: vegyük hozzá a tesztkészlethez a $t_3 = \langle 0, 1, 2, 3, 4, 5, ... \rangle$ és $t_4 = \langle 1, 2, 3, 4, 5, 6, ... \rangle$ input szekvenciákat mint további teszteseteket! Detektálunk-e hibát? Hogyan változnak a tesztfedési számok?

m. *Kiegészítő feladat: határozzuk meg, hogy pontosan milyen input szekvenciák esetén sérül R3, és javasoljunk hibajavítást!

Megoldás

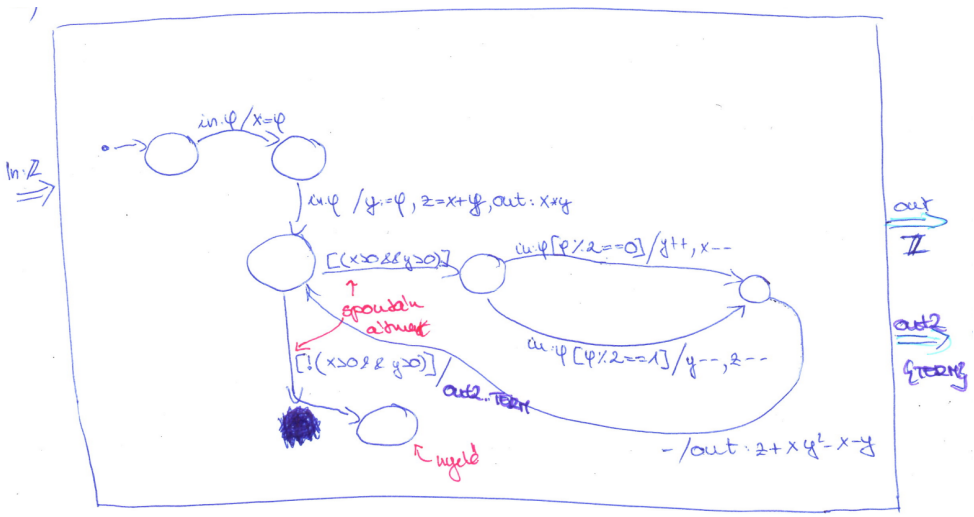
a. Készítsük el a folyamatmodellt. (GL: ezt ki kéne javítani, a feltételkiértékelés readInput nem szerencsés labelként, mert érthető úgy is, hogy nem mindig hajtódik végre. Legyen inkább explicit task.)



b. A folyamatmodellen jól látszik: minden út átmegy az első output adási tevékenységen.

c. Pozitív x és y esetén vagyunk a ciklusban; itt x nem nőhet, ezért a második ág csak véges sokszor hajtódhat végre; így pedig egy idő után csak az első ágat hajthatnánk végre, ahol y előbb-utóbb elfogy. (Van még az az eset is, hogy y negatívba túlcsordul, de akkor is rögtön leáll.)

d. Tanultunk állapotváltozókat, legyen ilyen az x, y, z. Írjunk az átmeneti élekre őrfeltételeket ezek segítségével, meg akciókat (mint a Yakinduban is). Lehet minden programsorra egy állapot, de lehet sokkal kevesebb is, pl. az egész ciklusmag lehet két hurokél egyazon állapot fölött (kicsit szebb annyira szébontani, hogy először a ciklusfeltételt teszteljük, utána a belső elágazást); ezek mind helyes megvalósítások lesznek.



e. Mit csinál a program? $t_1 = \langle 2, 3, 5, 7, 11, 13, ... \rangle$

- $x = 2$

- $y = 3$
- $z = 5$
- out: 6
- belépünk a ciklusba
- true ág
- $y = 2$
- $z = 4$
- out: $4 + 8 - 2 - 2 = 8$
- bent maradunk a ciklusban
- true ág
- $y = 1$
- $z = 3$
- out: $3 + 2 - 2 - 1 = 2$
- bent maradunk a ciklusban
- true ág
- $y = 0$
- $z = 2$
- out: $0 + 2 - 2 - 0 = 0$
- kilépünk a ciklusból
- futás vége
- Az orákulumon az outputok és a terminálás alapján végigjátszható, hogy (az elvárásoknak megfelelően) "elfogad" outputot ad ki.
- A két tesztet együtt nézve, a program összes utasítását végrehajtottuk, ez 100% utasításfedés. A folyamatmodellen karikázva elvileg azt látjuk, hogy minden csomópontban jártunk, az állapotgépnek is minden élét bejártuk.

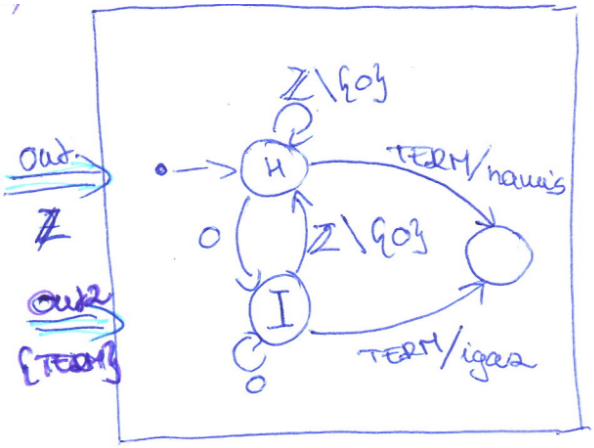
f. 9 utasítást hajt végre és 2 utasítást kihagy –> $9/11 \sim 82\%$-os utasításfedés. A vezérlési folyamon a csomópontokat lehet karikázni, és a csomópont szintű fedettséget jelenti (kis eltéréssel / korrekcióval, mert a decision-merge pár csak egyszer számított utasításnak).

g. $t_2 = \langle 1, 2, 4, 1, 2, 4, ... \rangle$ Mit csinál a program?

- $x = 1$
- $y = 2$
- $z = 3$
- out: 2
- belépünk a ciklusba
- false ág
- $x = 0$
- $y = 3$
- out: $3 + 0 - 0 - 3 = 0$
- kilépünk a ciklusból.
- futás vége
- Tehát nem sérült az R3.

h. Az állapotgépen fedettség szempontjából hasonló a helyzet, mint a kódban, amennyiben elég közeli formában építettük fel – de a javasolt összevonások után jóval egyszerűbb, nem látszik rajta ez a fedettségi mérték (van persze állapotfedettség, de az most akár 100% is lehet; megfelelő összevonásokkal itt csak az átmenetfedettség marad el a 100%-tól).

i. Teszt orákulum (a tesztbemenetet és a valós kimenetet alapján eldönti a teszt eredményét és I/H-t ad vissza). Az automata két legfontosabb állapota, hogy "utolsó input 0" és "utolsó input nem 0", ezek között értelemszerűen ugrál. A SUT terminálódásának hírére pedig átmegy a nyelő állapotba, és rendre "elfogad", ill. "elutasít" kimenetet ad. (A kezdőállapot működése nincs teljesen specifikálva, lehet mondjuk a "utolsó input nem 0" állapot). Az orákulumon az outputok és a terminálás alapján végigjátszható, hogy (az elvárásoknak megfelelően) "elfogad" outputot ad ki (TODO:elfogad/elutasít és igaz/hamis nem konzisztens a magyarázatban és a minta megoldásban).



j. Ha a folyamatmodell vezérlési éleit, ill. az állapotgép átmeneteit is jelöljük, akkor azt láthatjuk, hogy azokban is jártunk mind. Ehhez tartozik egy élfedési metrika, ami a programok esetén ágfedést jelent – most ez is 100%.

k. Egy lehetséges teszteset: $t_3 = \langle -1, -1, -1, ... \rangle$. Mit csinál a program?

- $x = -1$
- $y = -1$
- $z = -2$
- out: 1
- nem lépünk be a ciklusba
- futás vége
- ez bizony megsérti a követelményt, az orákulum fülön is csípi.
- Nem elég az utasítás vagy ág szintű fedettséget nézni, a *robusztusságot* is meg kell vizsgálni a szélső vagy kritikus inputértékekre és közvetlen környezetükben (jelen esetben ilyen a 0 az x és y esetén). Ez nagyon hasznos lesz a programozásos házi feladatoknál!

l. Otthoni munka.

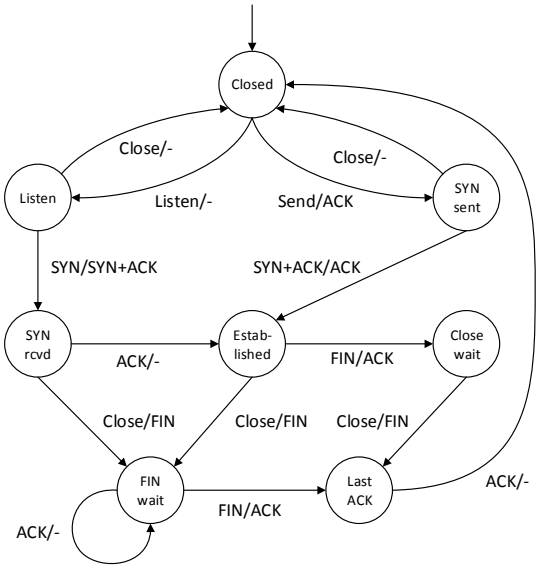
m. Lássuk csak!

- triviális megoldás a `void f() { writeOutput(0); }`, de inkább olyan megoldást keressünk, ami legalább nyomokban őrzi az eredeti működést.
- ha x és y is negatív, és nem lépünk be a ciklusba, akkor $x \cdot y$ lenne az utolsó output – ilyenkor el kell dönteni, mi a helyes viselkedés, pl. a ciklus helyett mondjuk ki lehetne adni egy 0 outputot. Vagy eleve elutasítani a negatív bemenetet, ha az érvénytelennek számít.

- $z = x + y$ végig igaz, így a ciklusmagban kiadott outputból egyedül a szorzattag marad.
- ha belépünk legalább egyszer a ciklusba, akkor a ciklus utolsó lefutása után, feltéve ha nem volt negatívba overflow, akkor x vagy y 0 lett, tehát a szorzatuk is 0, ez eleve stimmel.
- azt kell tehát még kivédeni, hogy ne overflowoljon y `MAXINT`-ból negatívba. eldöntendő, hogy ilyenkor mi a teendő, pl. egy magas küszöbszám feletti y értékek esetén hibát adunk és leállunk, vagy detektáljuk az overflow-t és kiadunk 0-t stb.

2. feladat

Az alábbi állapotgép a TCP protokoll egy részének működését írja le. (A Transmission Control Protocol (TCP) az internetes forgalomban fontos TCP/IP protokollcsalád egyik fő protokollja.) A TCP egy számítógépen futó program és egy másik számítógépen futó másik program között egy adatfolyam megbízható, sorrendhelyes átvitelét hivatott biztosítani. A protokoll egy jól elkülönülő része a kommunikációs csatorna felépítése és lebontása, ennek folyamatát mutatja be az állapotgép. A protokollnak adható parancsok a Listen, a Send és a Close. A küldött és fogadott üzenetek a SYN, az ACK és a FIN (az ábrán a SYN+ACK egy két parancsot tartalmazó üzenet). *Tipp:* a feladat megoldásához nem feltétlenül szükséges a protokoll működésének megértése.



a. Ellenőrizze és indokolja, hogy az állapotgép teljesen specifikált és determinisztikus-e! Adjon meg két lehetséges módot, ahogyan a nem teljesen specifikált állapotgépek esetén a fel nem tüntetett bemeneteket kezelhetjük!
b. Adjon egyetlen, legfeljebb 10 bemenetből álló tesztesetet, ami 100%-os állapotfedést eredményez! Adja meg a tesztesetre az állapotgép által adott kimenetsorozatot is! Tipp: a feladat 8 bemenettel is megoldható.
c. Az eredeti állapotgépből kiindulva finomítással bontsa ketté a „FIN wait” állapotot. Rajzolja fel azt a variációt, amelyikben a finomított állapotgép pontosan egy ACK üzenetet fogad a Close parancs és a FIN üzenet fogadása között!
d. Absztrakcióval vonja össze a „SYN rcvd” és „Established” állapotokat az „Abs. state” állapotba! (1p)
e. A b. feladatban elkészített tesztesetet a finomított (c.) / absztrahált (d.) modellen lejátszva (az esetleges nemdeterminisztikus döntéseknél megfelelően választva) megkaphatjuk-e a korábbi kimeneteket? Általánosan is érvényes-e ez a megállapítás, ha tetszőleges (de szabályos) finomítást alkalmazunk egy állapotgép állapotain és újrafuttatunk egy korábbi tesztkészletet? A válaszokat indokolja!

Megoldás

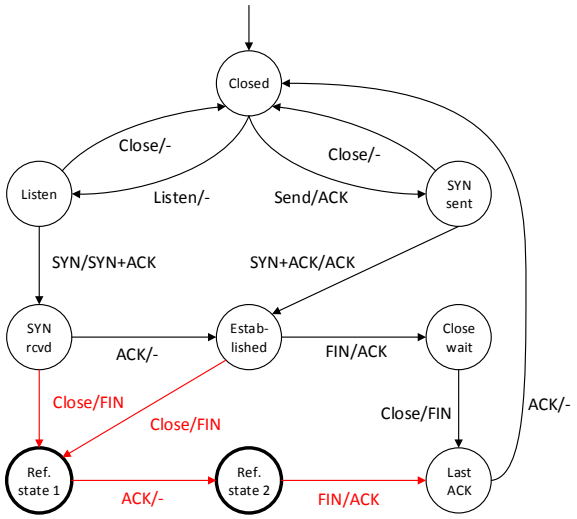
a.

- Determinisztikus, mert semelyik állapotból nem indul ki két, azonos bemenetű átmenet. (1p)
- Nem teljes, mert pl. a „Listen” állapotban már nem kezelt a „Listen” parancs. (1p)
- Jó válaszok: 1) kimenet nélküli hurokélként kezeljük, tehát a modell elnyeli, nem reagál rá; 2) tiltott bemenetként tekintünk rájuk és érkezésükkor hibás állapotba kerül a rendszer; 3) nem specifikált a viselkedés, bármi történhet (tehát feltételezzük, hogy ilyen soha nem történik). Ezek közt a döntés lehet domain specifikus, ld pl kritikus beágyazott rendszereknél követelmény a teljes specifikáció, Yakindu eldobja az eseményt. (0,5p+0,5p)

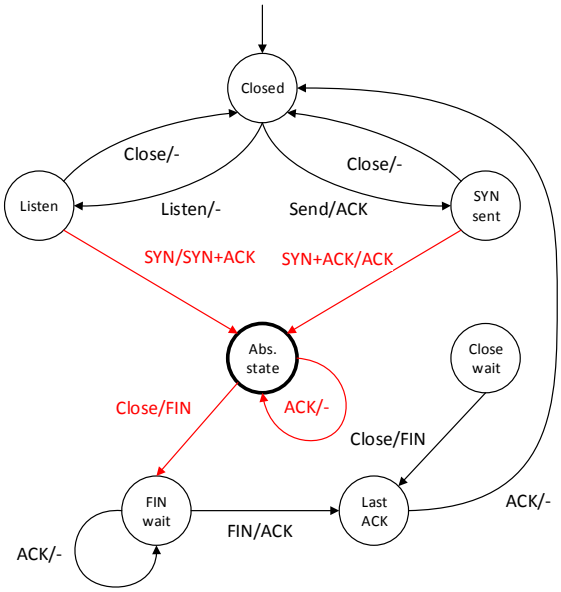
b.

- Javításkor célszerű lejátszani a hallgató által adott bemenetsorozatokat az állapotgépen, és piros tollal jelölni az érintett állapotokat (valószínűleg ők is így csinálják majd). A pontozás hiba-alapú, tehát (amennyiben szerepel megoldás) 4 pontról indul, és minden hibáért levonás jár. Lehetséges hibák:
- Nem stimmel egy-két kimenet, de alapvetően jó (-0,5p)
- Egyáltalán nincsenek kimenetek, vagy teljesen rosszak (-1p)
- A teszteset nem lejátszható, de láthatóan csak elnézett egy bemenetet (-1p)
- Több helyen is elnézte, de vannak arra utaló jelek, hogy tudta mit kell csinálni (-2p)
- Egyáltalán nem lejátszható tesztesetek (az egész feladat 0p)
- Kimaradt egy állapot a fedésből (-1p)
- Több állapot is kimaradt a fedésből (-2p)
- A teszteset hosszabb, mint 10 bemenet (-1p)

c.



d.



e.
Jó eséllyel nem lesz lejátszható a teszteset, mert állapotfedéskor nem túl okos dolog hurokélet használni. Ha semmi mást nem csinált, de van nyoma annak, hogy ezt ellenőrizte (tehát nem csak mondja), az 1p, akkor is, ha a c) feladat nem lett hibátlan. A második résznél alapvetően kétféle válaszra kell számítani. Az egyik az, hogy tesztesetek általában nem mindig játszhatók le. Erre indoklás nélkül megint nem jár pont. Megfelelő indoklások lehetnek a következők:

- Állapotok szétbontásakor hurokélekből két állapot közötti él keletkezhet, így használata már nem opcionális, tehát minden olyan teszteset lejátszhatatlanná válik, ami az állapoton a hurokél használata nélkül haladt át.
- Állapotok szétbontásakor az elérhetőséget is változtatni lehet, mivel a bemenő éleket az egyik állapotra, a kimenőket a másikra, a hurokéleket pedig hurokélként megtartva a két új állapot között nem lehet áthaladni. Így minden olyan teszteset végrehajthatatlanná válik, ami áthaladt ezen az állapoton.
- Finomítás során letilthatóak eddig megengedett viselkedések/csökkenhet a lehetséges megvalósítások száma, így a tesztesetek nem feltétlenül lesznek lejátszhatók.
- Ellenpélda a most megadott nem lejátszható teszteset.

A másik az, hogy a tesztesetek lejátszhatók maradhatnak (szigorúan feltételes módban), annak ellenére, hogy most nem voltak lejátszhatók (ez alapvetően a feladat kismértékű félreértése, ezért 1 pontot levonunk érte, de védhető). Erre lehetséges indoklások:

- Nem biztos, hogy a tesztkészlet egyáltalán érintette ezt az állapotot, tehát lehet, hogy lejátszható maradt.
- Ha a teszteset érinti a hurokélet is az állapoton (1p), és a finomítás során a keletkező két állapoton át vezet út (2p), akkor lejátszható marad.
- A jó indoklásra automatikusan 4 pont jár (a második válasz esetén 3), függetlenül attól, hogy megpróbálta-e lejátszani a teszteseteket. Közelítő magyarázatokra (tudja, hogy a finomítás micsoda, tudja, hogy elveszhet viselkedés, stb.) érzés szerint részpontszám adható.