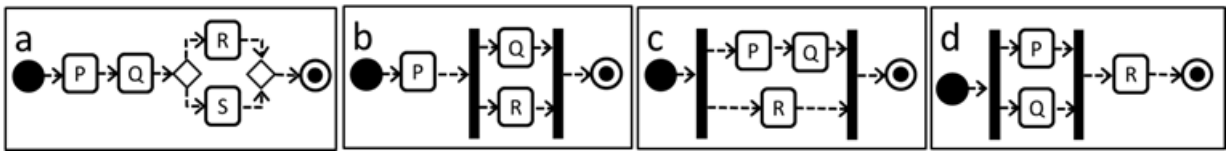
3. gyakorlat – Folyamatmodellek, kooperáló viselkedésmodellek – Megoldások

Figyelem: Jelen anyag belső használatra készült megoldási útmutató, melyet a ZH felkészülés segítése érdekében publikáltunk. A feladatok részletesebb megoldása magyarázattal gyakorlaton hangzott el.

1. feladat

Egy folyamat végrehajtása során az összes lépést megfigyeltük. A következő eseménysor bekövetkeztét észleltük:
Folyamat indul, *P elkezdődik*, *P befejeződik*, *Q elkezdődik*, *R elkezdődik*, *Q befejeződik*, *R befejeződik*, *Folyamat befejeződik*.



Az a, b, c, d folyamatmodellek közül melyek lehetnek helyes modelljei a rendszernek?

Megoldás

b és *c*

2. feladat

Felhő alapú adattárolást modellezünk (ld. Dropbox, Google Drive, Tresorit), egyetlen állományra szorítkozva. Az állománynak a szerveren és a kliensnél (pl. laptop) is elérhető egy-egy replikája, kezdetben azonos tartalommal. A fájl módosításai *szinkronizálás* során továbbítódnak a példányok között. Ha szinkronizáció előtt mindkét példányt módosítják, akkor *ütközés* lép fel, amelyet a felhasználónak kell feloldania a kliensen.

a. Modellezzük először a laptop kliens (részleges) működését állapotgéppel! A kliens kezdetben *szinkron* állapotú (a lokális fájlmásolat egyezik azzal, ami a szerveren a legutóbbi szinkronizációkor volt/lett), ám *írás* input hatására a *piszkos* állapotba kerül (és további *írás* hatására is ottmarad). Az *elvet* input hatására tetszőleges állapotból újra *szinkron* állapotba kerül.
b. A szerver lehetséges állapotai (csupán az adott laptop klienssel való szinkronizációt vizsgálva) a *szinkron* és a *frissült*. Előfordulhat, hogy a megfelelő írási jog birtokában egy másik felhasználó (vagy ugyanazon felhasználó egy másik kliens, pl. a telefonja segítségével) frissíti a szerveren található állományt.
c. Ha a szerver *szinkron* állapotban van, akkor a kliens a *szinkronizál* input hatására feltölti az esetleges lokális módosításokat a szerverre, és szintén *szinkron* állapotba kerül. A kliens és a szerver közben információt cserélnek – hol kooperál a két automata?
d. Ha a szerver *frissült* állapotban van, akkor a kliensnek adott *szinkronizál* input hatására a szerver *szinkron* állapotba kerül; a kliens pedig *szinkron* állapotból nem mozdul, de *piszkos* állapotból *ütközés* állapotba megy. Mit jelent ez? Mi történjen az ütközés állapotban? Hol kooperál a két automata?
e. A kliens időnként magától is szinkronizál a szerverrel, felhasználói input nélkül. Mit jelent ez? Hol kooperál a két automata
f. (Otthoni feladat) Teljes összetett állapottér kifejtése a vegyes szorzatban részvevő két automata alapján.

Megoldás

a. Megoldás:

b. Itt annyi az érdekesség, hogy spontán állapotátmenet lesz (már ha a lokális usertől érkezett inputokra figyelünk csak). c. A Kliens automata két új átmenetére `[Szerver.szinkron]` őrfeltételt rakunk (az őrfeltétel fogalma volt előadáson!). Érdekesség, hogy ez a másik automata pillanatnyi állapotától függ - ez tehát kooperáció! A modell absztrahálja a valóságot, itt ténylegesen nyilván üzenetcsere van a kliens és a szerver között, ahol kicserélik ezt az információt, ill. a fájlt is fel kell tölteni. d. Itt egyszerre lép a két automata, tehát ez egy olyan input, amelyik mindkét automatát befolyásolja! Ilyenkor azt mondjuk, hogy nem aszinkron, hanem vegyes szorzatban van a két komponenes automata (főleg aszinkron lépnek, de néha szinkron, mert mindkettő egyszerre lép). A kliens új élei `[Szerver.frissült]` őrfeltételt kapnak. Megjegyzés: a Kliens.szinkron állapotból két hasonló átmenet megy ki ellentétes őrfeltétellel (`[Szerver.szinkron]`, ill. `[Szerver.frissült]`), ezek összevonhatóak egyetlen őrfeltétel nélküli éllé. (Ne felejtsük el, hogy a Szerver új éleit berajzoljuk a megfelelő őrfeltétellel.) Ez nyilván az az eset, amikor a kliens detektálja az ütközést. Ilyenkor a szerver szinkron állapotba megy, mivel őnála a legutolsó szinkronizáció óta nincs újabb. A kliensen kell feloldani a konfliktust, pl. így: elvet->szinkron, írás->piszkos, szinkronizál->ütközés). e. Ugyanaz, mint előbb, csak külső input esemény helyett közös, belső randevú esemény (legyen *auto*) hatására. Opcionális változtatás lehet, pl. konfliktust ilyenkor sose idézzen elő.

3. feladat

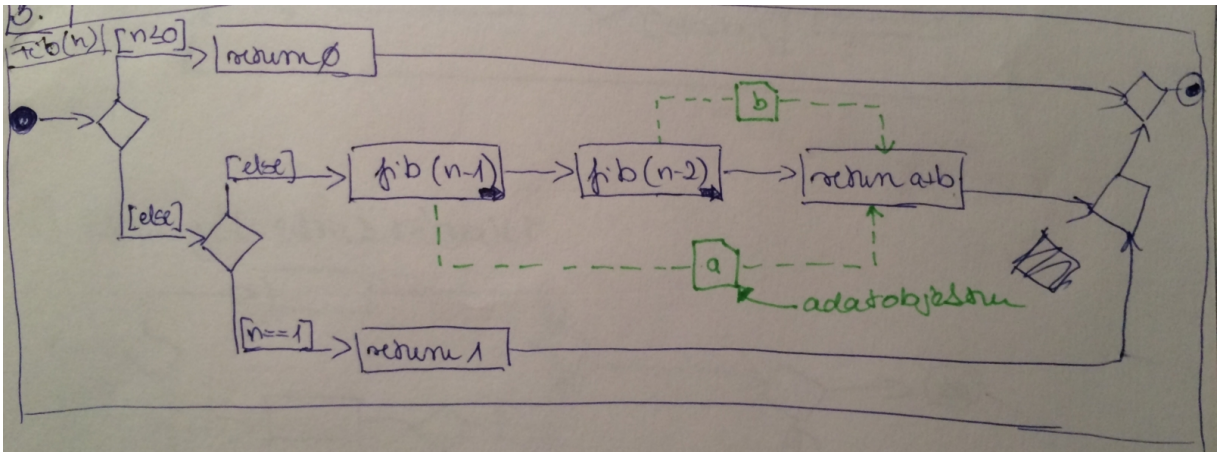
Tekintsük az alábbi C nyelvű függvényt.

```
unsigned long long f(int n) {
   if (n <= 0) {
      return 0;
   } else if (n == 1) {
      return 1;
   } else {
      unsigned long long a = f(n - 1);
      unsigned long long b = f(n - 2);
      return a + b;
   }
}
```

a. Milyen vezérlési folyamot határoz meg a függvény?
b. Mi biztosítja azt, hogy a függvény előbb-utóbb terminál?
c. Azonosítsuk az adatfüggőségeket (adatáramlást) a tevékenységek között!
d. (Kiegészítő feladat) Ha a programozási nyelv vagy a futtatókörnyezet megengedi, hol van lehetőség párhuzamosításra?

Megoldás

a. A rekurzív hívás egy “hívás” elem.



b. Aki nem tudja, gondolkodjon rajta otthon. c. A két rekurzív hívásból megy adatáramlás az összeg returnjébe. A lényeg, hogy a második híváshoz voltaképp nem kell az első eredménye. d. A fenti alapján ez triviális.

4. feladat

a) Belsős fejlesztők közvetlenül írhatnak a kódtár adott projekt részére fenntartott területére. Külsős fejlesztőknek először átvizsgálásra (*code review*) be kell nyújtaniuk a kódjukat; ezután egy belső fejlesztőnek ellenőriznie kell azt, és utána vagy elutasítania, vagy elfogadnia. Az elfogadást követően az alapítvány jogi osztálya egy külön adminisztratív eljárásban tisztázza a változtatások szellemi tulajdonának jogállását, és csak ennek sikeres lezárása után olvaszthatja be a belső fejlesztő a kódot. Frissen indított, első hivatalos kiadásuk előtt álló projekteknél itt tesznek egy kivételt: az elfogadott külső hozzájárulás kódtárba beolvasztásával nem kell megvárni ezt az adminisztratív eljárást. Készítsünk folyamatmodellt az itt leírt tevékenységekből!

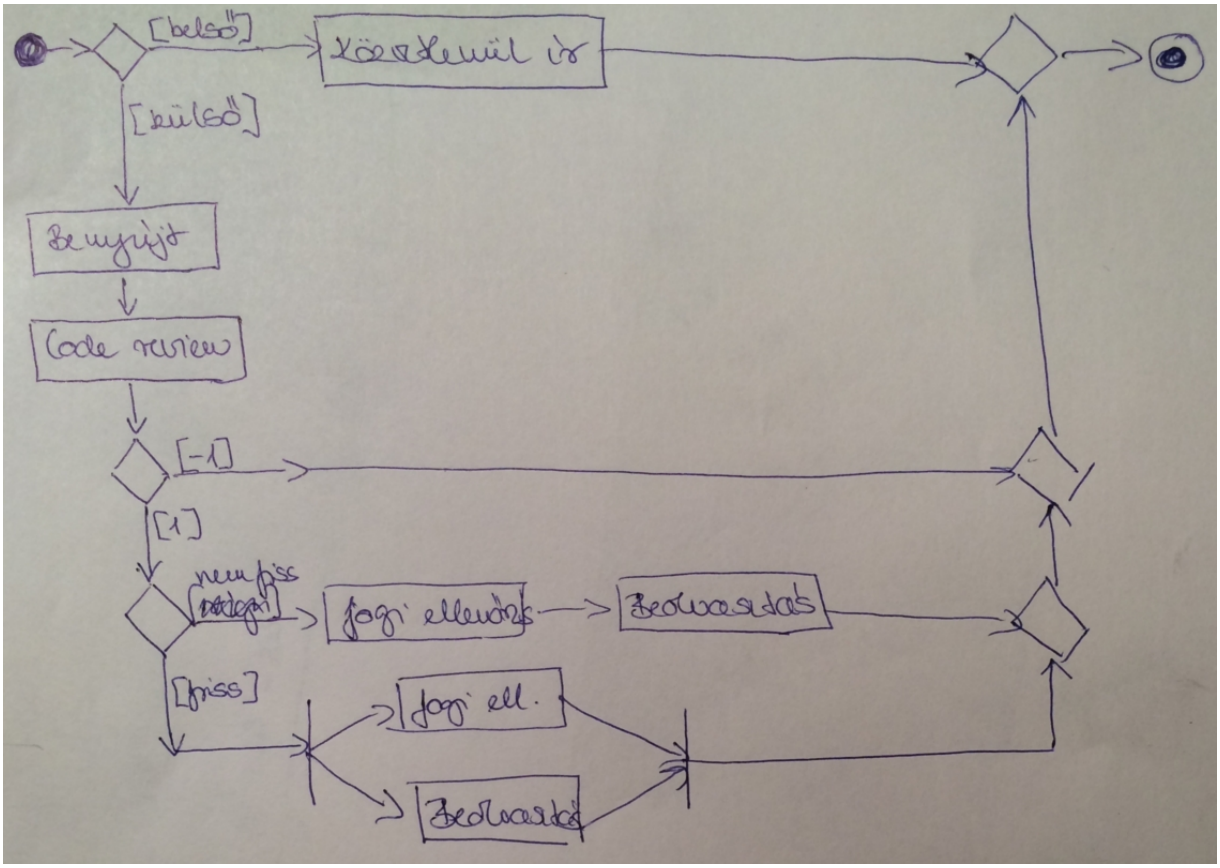
b) A szoftver fejlesztési projektje abból áll, hogy újabb és újabb módosításokat végeznek a forráskódon, amíg a projekt vezetése úgy nem látja, hogy a szoftver kellően stabil egy hivatalos kiadáshoz. Amikor eljött ez a pont, akkor közzétesznek egy új stabil verziót a szoftverből, majd ismét a fejlesztésen a sor stb. Készítsünk folyamatmodellt az itt leírt tevékenységekből!

c) Milyen viszonyban állnak egymással a fenti részfeladatokban elkészített folyamatmodellek?

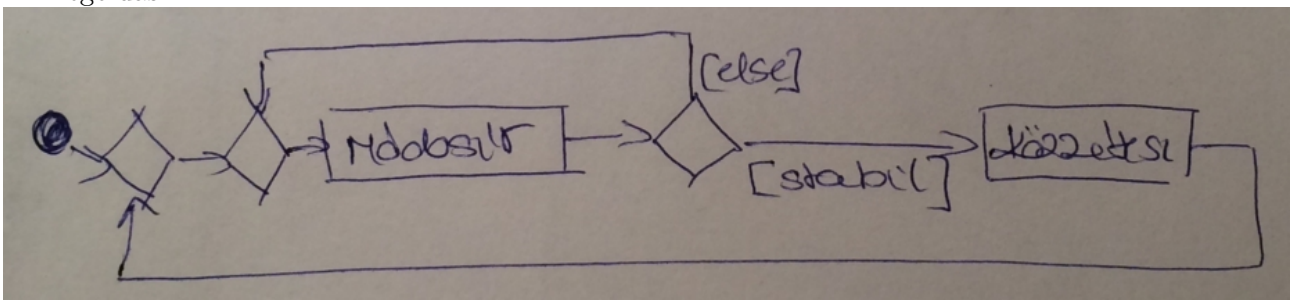
d) Ellenőrizzük, hogy jólstrukturáltak-e a folyamataink!

Megoldás

a. A megoldás:



b. A megoldás:



c. Két külön dolognak az életútja egyazon rendszerben (lehetnek furcsa átlapolódások, pl. kiadási cikluson átívelő code review).

d. Lásd a jólstrukturáltságról szóló anyag: http://docs.inf.mit.bme.hu/remo/out/04-folyamatmodellezes.html

5. feladat

Az első feladatban modellezett klienst és szervert alaposabb vizsgálatnak vetjük alá, figyelembe véve, hogy hálózaton keresztül, üzenetek segítségével tartják a kapcsolatot. Ami az előző, absztrakt modellben a két automata egyidejű (atomi) közös állapotátmeneteként jelent meg, az valójában időbeli kiterjedéssel rendelkező, üzenetek küldésével és fogadásával megvalósuló kommunikációs tevékenység.

a. Készítsünk adatfolyamhálót, amelynek a kliens és a szerver a csomópontjai! Az adatfolyamhálóban FIFO, kapacitáskorlát nélküli, pont-pont csatornákat használunk (az előadáson bevezetett módon). Határozzuk meg az adatfolyamháló bemenetén, illetve a két csomópont közötti interfészen érvényes tokeneket és a jelentésüket!
b. Adjuk meg a csomópontok belső viselkedésének egy lehetséges modelljét állapotgráf formalizmussal! Az állapotgráfot írjuk fel táblázatos formában is.
c. Hogyan modellezhető az üzenetek belső strukturája?

Megoldás

a. A felhasználói input ugyanaz. A kliens és a szerver között megy oda egy kérés, vissza egy válasz (első körben). Érdemes megkérdezni, hogy értik-e miért nem közvetlenül a felhasználó *szinkronizálj* üzenete jut el a szerverre, és hogy világos-e, mikor kommunikálna a két komponens.

b. A szerver értelemszerű. Kliens: minden őrfeltétel vagy szinkron lépés esetén először egy kérést küld, majd várakozó állapotba megy. A várakozó állapotból a válasz beérkezésekor lép ki (és ha közben semmi mást nem csinál, akkor visszaabsztrahálva valóban pillanatszerű, atomi szinkron művelet lesz). Ha őrfeltétel volt, az új modellben hova lép ki a válasz beérkeztekor? Hoppá, ez nem derül ki → finomítsuk a válasz tokent! Lesz `válaszSZ` (ha a szerver szinkronban van) és `válaszF` (ha a szerver frissítve állapotban volt), ez alapján a kliens már egyértelműen tud dönteni, de persze a szervert hozzá kell igazítani. (Nem lesz idő az egész klienst lerajzolni, a befejezése otthoni munka!)üü

c. Itt egy rövid pár mondatos utalást lehet tenni, hogy az előző tokenfinomítás megfelel a *strukturális modellezés* előadásban használt altípus fogalomnak; az ott tanult eszköztár egyéb elemei is hasznosak lehetnek, pl. a kérés/válasz üzeneteknek lehet egy verziószám attribútuma, meg tartalmazhatnak fájlt.