Tartalomjegyzék

I. SZINKRON HÁLÓZATI ALGORITMUSOK

A könyv első része a 2. fejezettől a 7. fejezetig tart. A fejezetekben algoritmusok és alsó korlátok találhatók, ezek olyan *szinkron hálózati modellre* vonatkoznak, amelyben a hálózati processzorok szinkron menetekben lépnek és cserélnek üzeneteket.

Ennek a résznek az első fejezete, a 2. fejezet, a szinkron hálózatokra megadott formális modellünket írja le. Az olvasó átugorhatja ezt a fejezetet, és ha szükséges, visszatérhet ide akkor, amikor az algoritmusokkal foglalkozó 3–7. fejezeteket olvassa. A 3. fejezet azzal az egyszerű problémával foglalkozik, hogyan történik *egyetlen vezető kiválasztása egy gyűrűs hálózatban.* A 4. fejezet a tetszőleges gráfú szinkron hálózatokban használt alapalgoritmusokat tekinti át. Az 5. és 6. fejezet azokkal az alapproblémákkal foglalkozik, hogyan lehet vonalhibák és processzorhibák esetén a szinkron hálózatokban *megegyezést elérni.* Végül, a 7. fejezet az alapvető megegyezési problémák kiterjesztéseit és variációit tartalmazza.

2. fejezet

Modellezés/I. Szinkron hálózati modell

Ez a könyv legrövidebb fejezete. Mindössze egy egyszerű kiszámítási modellt ad a szinkron hálózati algoritmusokra. A modellt elkülönítve adjuk meg, úgy, hogy ezt a fejezetet könnyen használhassuk majd a 3–7. fejezet olvasásakor is.

2.1.. Szinkron hálózati rendszerek

A *szinkron hálózati rendszer* egy irányított hálózati gráf csúcsaiban elhelyezett számítási elemekből áll. Az 1. fejezetben ezek a számítási elemek „processzorok" voltak, amiből arra lehet következtetni, hogy ezek valamilyen hardver berendezések. Ezeket azonban gyakran hasznosabb olyan logikai szoftver „folyamatoknak" tekinteni, amelyek a processzorokon futnak (de nem azonosak a processzorokkal). A bemutatandó eredmények mindkét értelmezésre érvényesek. A könyvben mostantól kezdve a számítási elemeket „folyamatoknak" fogjuk nevezni.

A szinkron hálózati rendszer formális meghatározását egy $G = (V, E)$ irányított gráffal kezdjük. Jelöljük n-nel $|V|$-t, a hálózati irányított gráf csúcsainak számát. G minden P_i csúcsára jelöljük $\mathit{ki\_szomszédok}_i$-vel P_i „kimenő szomszédait", azaz azokat a csúcsokat, amelyekbe a P_i-ből irányított élek vezetnek, és $\mathit{be\_szomszédok}_i$-vel P_i „bemenő szomszédait", azaz azokat a csúcsokat, amelyekből él vezet P_i-be. Jelöljük $\mathit{távolság}(i, j)$-vel a P_i-ből a P_j-be vezető G-beli legrövidebb út hosszát; ha ilyen út nem létezik, akkor legyen $\mathit{távolság}(i, j) = \infty$. Határozzuk meg *átm*-ot, az *átmérőt* úgy, hogy minden (i, j) párra meghatározzuk a $\mathit{távolság}(i, j)$-k maximumát.

Azt is feltesszük, hogy adott egy rögzített M ábécé, az üzenetek ábécéje, és *null* jelölje azt, hogy nincs üzenet.

Minden $i \in V$ csúcshoz hozzárendelünk egy olyan *folyamatot*, amely formálisan a következő összetevőkből áll:

- $\mathit{állapotok}_i$, az *állapotok* (nem feltétlenül véges) halmaza;
- $\mathit{kezdő}_i$, az $\mathit{állapotok}_i$ nemüres részhalmaza, a *kezdeti állapotok* vagy *kezdő állapotok* halmaza;

- *üzenetek$_i$*, az *üzenetgeneráló függvény*, ami az *állapotok$_i$* $\times$ *ki_ szomszédok$_i$*-t leképezi $M \cup \{null\}$-ra;
- *átmenetek$_i$*, *állapotátmeneti függvény*, ami az *állapotok$_i$*-t és az $M \cup \{null\}$ elemeinek vektorait (amelyeket a *be_ szomszédok$_i$*-vel indexelünk) leképezi *állapotok$_i$*-re.

Tehát minden folyamatnak van egy állapothalmaza, az állapothalmaz egy részhalmazának pedig kitüntetett szerepe van, ez a kezdő állapotok részhalmaza. Az állapotok halmaza nem feltétlenül véges. Ez fontos, mivel lehetővé teszi azt, hogy olyan rendszereket modellezzünk, amelyekben nemkorlátos adatszerkezetek, például számlálók is vannak. Az üzenetgeneráló függvény minden állapothoz és kimenő szomszédhoz megadja az üzenetet (vagy nem ad üzenetet), ezt a P_i folyamat elküldi az adott állapotból induló megadott szomszédhoz. Az állapotátmeneti függvény minden állapotra és minden bemenő szomszédtól jövő üzenethalmazra megadja azt az új állapotot, amelyikbe a P_i folyamat megy.

A G minden (i, j) éléhez adott egy *csatorna*, amit *vonalnak* is nevezhetünk, ez lényegében egy olyan tároló, amely minden időpillanatban legfeljebb egy M-beli üzenetet tud tárolni.

Az egész rendszer végrehajtása úgy indul, hogy minden folyamat egy tetszőleges kezdő állapotban van és a csatornák üresek. Ezután a folyamatok óraütemenként a következő két lépés végrehajtását ismétlik.

1. A pillanatnyi állapotra alkalmazni kell az üzenetgeneráló függvényt, ez hozza létre az össszes kimenő szomszédhoz küldendő üzenetet. Az üzeneteket a megfelelő csatornákra kell helyezni.
2. Az állapotátmeneti függvényt a pillanatnyi állapotra és a bejövő üzenetekre kell alkalmazni, a függvény megadja az új állapotot. A csatornákról el kell távolítani minden üzenetet.

A két lépés kombinációját *menetnek* nevezzük. Megjegyezzük, hogy – általában – nem teszünk semmilyen korlátozást azoknak a számításoknak a méretére, amiket a folyamat végez akkor, amikor meghatározza az üzenetgeneráló és az állapotátmeneti függvények értékeit. Azt is megjegyezzük, hogy az itt bemutatott modell determinisztikus, abban az értelemben, hogy az üzenetgeneráló függvény és az állapotátmeneti függvény (egyértékű) függvények. Így, ha adott a kezdő állapotok egy halmaza, akkor a végrehajtás csak egy úton haladhat.

Megállás.. Eddig még egyáltalán nem beszéltünk a *folyamatok megállásáról*. Könnyű elkülöníteni néhány folyamat-állapotot, mint *megállító állapotot*, és azt mondani, hogy ezekből az állapotokból semmilyen tevékenység nem indul, azaz semmilyen üzenet nem jön létre és az állapotátmenet eredménye is az eredeti állapot. Megjegyezzük, hogy ezek a megállító állapotok nem azt a szerepet játsszák, mint a hagyományos véges állapotú automaták ilyen állapotai. Azok általában mint *elfogadó állapotok* szerepelnek, azaz olyan állapotok, amelyek meghatározzák, hogy egy nyelv mely jelsorozatait számította ki a gép. Itt ezek az állapotok csak arra szolgálnak, hogy a folyamatokat megállítsák; azt, hogy egy folyamat mit számolt ki, más szabályoknak megfelelően kell meghatározni. Az elfogadó állapotok fogalmát az osztott algoritmusok rendszerint nem használják.

Különböző kezdési időpontok.. Időnként olyan szinkron rendszerekkel is foglalkozni akarunk, amelyben a folyamatok különböző menetekben kezdik a működésüket. Ezt az esetet úgy modellezhetjük, hogy a hálózati gráfot kiegészítjük egy olyan *környezeti csúccsal*, amelyikből él indul minden más csúcsba. A csúcshoz tartozó *környezeti folyamat* speciális *ébresztő* üzeneteket küld a többi folyamatnak. A többi folyamat minden kezdő állapota kezdetben *csendes*, ami azt jelenti, hogy semmilyen üzenetet nem hoz létre, és ezt a viselkedést csak akkor változtatja meg, ha a környezeti folyamattól *ébresztő* üzenetet kap, vagy ha valamelyik másik folyamat nem-*null* üzenetet küld neki. Tehát egy folyamat elindulhat közvetlenül, a környezeti folyamattól kapott *ébresztő* üzenet miatt, vagy közvetetten, egy már korábban elindult folyamat nem-*null* üzenete által.

Irányítatlan gráfok.. Néha azokkal az esetekkel is foglalkozni akarunk, ahol a hálózati gráf irányítatlan. Ezt az esetet az irányított gráfokra korábban már megadott modellel úgy írhatjuk le, hogy egy olyan irányított gráfhálózatot adunk meg, ahol bármely két szomszédos csúcs között kétirányú él van. Ebben az esetben P_i gráfbeli szomszédjainak jelölésére a $szomszédok_i$ jelölést használjuk.

2.2.. Hibák

Most a szinkron rendszerek különböző típusú hibáit fogjuk áttekinteni; foglalkozunk mind a *folyamathibákkal*, mind a *vonal- (csatorna-) hibákkal.*

A folyamat *megállási hibát* mutat akkor, ha egyszerűen megáll valahol a működése közben. Modellünk fogalmainak megfelelően a folyamat hibás lehet a fenti első vagy második lépés valamelyik végrehajtása előtt vagy után; sőt azt is megengedjük, hogy valahol az első lépés közepén is hibás lehet. Ez azt jelenti, hogy a folyamat csak az üzenetek egy részhalmazát tudja az üzenetcsatornákra helyezni. Feltesszük, hogy ez *tetszőleges* részhalmaz lehet – és nem gondoljuk azt, hogy a folyamat az üzeneteket sorban állítja elő, és hogy valahol a sor közepén hibásodik meg.

A folyamat *bizánci hibát* is mutathat: ekkor a folyamat önkényes módon hozza létre a következő üzeneteket és a következő állapotot, anélkül tehát, hogy a megadott üzenetgeneráló és állapotátmeneti függvényeket használná.

A vonal hibás lesz, ha az üzenetek elvesznek. Modellünk kifejezéseivel: a folyamat megpróbál egy üzenetet a csatornára helyezni az első lépés alatt, de a hibás vonal nem tudja rögzíteni az üzenetet.

2.3.. Bemenetek és kimenetek

Eddig még semmit sem szóltunk a bemenetek és kimenetek modellezéséről. Az állapotokban szereplő bemenetek és kimenetek kódolására egy egyszerű módszert használunk, mégpedig azt, hogy a bemenetek a kezdő állapotokban levő kijelölt bemeneti változókba kerülnek. Az, hogy egy folyamatnak több kezdő állapota lehet, itt most azért fontos, hogy összegyűjthessük a különböző lehetséges bemeneteket. Valóban, feltesszük, hogy a több kezdő állapot *csak* azért szükséges, hogy lehetséges legyen az, hogy különböző bemenetértékek legyenek a bemeneti változókban. A kimenő értékek a kijelölt kimeneti változókban vannak; ezek a változók csak a rájuk vonatkozó első írási művelet eredményét tárolják (azaz *egyszer írható* változók). Ugyanakkor a kimeneti változókat tetszőlegesen sokszor lehet kiolvasni.

2.4.. Végrehajtási sorozatok

A szinkron hálózati rendszer viselkedésének bemutatásához meg kell adnunk a rendszer „végrehajtásának” formális leírását.

A rendszer egy *állapot-hozzárendelését* úgy határozzuk meg, hogy a rendszer minden folyamatához egy állapotot rendelünk. Az *üzenet-hozzárendelés* azt jelenti, hogy minden csatornához hozzárendelünk egy üzenetet (esetleg a *null* üzenetet). Egy rendszer *végrehajtási sorozatát* a végtelen

$$C_0, M_1, N_1, C_1, M_2, N_2, C_2, \dots$$

sorozattal adjuk meg, ahol mindegyik C_r egy állapot-hozzárendelés és mindegyik M_r és N_r egy üzenet-hozzárendelés. C_r, M_r és N_r ($r = 0, 1, 2, \dots$) együtt alkotják a végrehajtási sorozat r-edik menetét: C_r a rendszer állapotát jelenti az r-edik menet után, míg M_r és N_r jelenti az r-edik menet alatt elküldött és kapott üzeneteket. (Ezek különbözők lehetnek, mivel a csatornákon az üzenet elveszhet.) A C_r-re gyakran úgy hivatkozunk, mint az *r időpontban* jelentkező állapot-hozzárendelésre, az r időpont az r-edik menet végrehajtása utáni időpontot jelzi.

Ha α és α' egy rendszer két végrehajtási sorozata, akkor azt mondjuk, hogy a P_i folyamat szempontjából az α *megkülönböztethetetlen* α'-től és ezt $\alpha \stackrel{i}{\sim} \alpha'$-vel jelöljük, ha α-ban és α'-ben P_i-nek ugyanaz az állapotsorozata, ugyanaz a kimenő üzeneteinek a sorozata, és ugyanaz a bemenő üzeneteinek a sorozata. Hasonlóan, azt mondjuk, hogy *a P_i folyamat szempontjából* α és α' *megkülönböztethetetlen az r-edik menetig*, ha α-ban és α'-ben P_i-nek az r-edik menet befejezéséig ugyanaz az állapotsorozata, ugyanaz a kimenő üzeneteinek a sorozata, és ugyanaz a bemenő üzeneteinek a sorozata. Ezeket a meghatározásokat kiterjesztjük azokra az esetekre is, ahol két különböző szinkron rendszer végrehajtási sorozatait hasonlítjuk össze.

2.5.. Bizonyítási módszerek

A szinkron rendszereknél használt legfontosabb bizonyítási módszer az *invariáns állítások* bizonyítása. Az invariáns állítás a rendszer állapotának (minden folyamat állapotának) olyan tulajdonsága, amely igaz minden végrehajtás, minden menet után. Megengedjük, hogy a befejezett menetek száma szerepeljen egy állításban, azért, hogy minden r-edik menet utáni állapotról állításokat mondhassunk. A szinkron rendszerekre vonatkozó invariáns állítások általában r szerinti teljes indukcióval bizonyíthatók, ahol r a befejezett menetek száma, és r kezdőértéke nulla.

Egy másik fontos módszer a *szimuláció*. Vázlatosan, a cél az, hogy megmutassuk, egy A szinkron algoritmus utánoz egy másik B szinkron algoritmust abban az értelemben, hogy ugyanazt a bemenő/kimenő működést mutatja. Az A és B közötti kapcsolatot olyan, az A és B állapotaira vonatkozó állítások fejezik ki, ahol a két algoritmus azonos bemeneti adatokkal indul és az azonos sorszámú menetekre azonos hibajelenségeket ad. Az ilyen állítást *szimulációs relációnak* nevezzük. Az invariáns állításokhoz hasonlóan a szimulációs relációkat is a befejezett menetek száma szerinti teljes indukcióval bizonyítjuk.

2.6.. Bonyolultsági mértékek

A szinkron elosztott algoritmusokra két bonyolultsági mértéket szokás használni, az időbonyolultságot és a kommunikációs bonyolultságot.

A szinkron rendszerek *időbonyolultságát* a végrehajtott meneteknek az a száma adja, ameddig a kívánt kimenetek megjelennek vagy amikor minden folyamat megáll. Ha a rendszer lehetővé teszi a különböző kezdési időpontokat, a bonyolultság attól az első menettől kezdve számítódik, amelyben minden folyamat *felébredt*.

A *kommunikációs bonyolultság* jellemzően az elküldött nem-*null* üzenetek számát jelenti. Esetenként az elküldött üzenetek bitszámát is figyelembe fogjuk venni.

A gyakorlatban az időmérték a fontosabb mérték, nem csak a szinkron osztott algoritmusok esetében, hanem minden osztott algoritmusnál. A kommunikációs bonyolultság főleg akkor fontos, ha olyan forgalomtorlódást okoz, ami lelassítja a végrehajtást. Emiatt mi nem foglalkozunk ezzel, csak az időbonyolultságot vizsgáljuk. A kommunikációs terhelésnek az időbonyolultságra való hatása azonban nem csak egyetlen osztott algoritmustól függ. A tipikus hálózatokban sok osztott algoritmus fut egyidejűleg, megosztva ugyanazt a hálózati sávszélességet. Egy algoritmus által adott vonalra küldött üzenet megnöveli ezen a vonalon a teljes üzenetterhelést, így hozzájárul ahhoz a forgalomtorlódáshoz, amit mindegyik algoritmus lát. Mivel nehéz kifejezni azt a hatást, amit az egyes algoritmusok üzenetei egy másik algoritmus időteljesítményére gyakorolnak, egyszerűen az egyes algoritmusok által létrehozott üzenetek darabszámát elemezzük (és megkíséreljük a darabszám minimalizálását).

2.7.. Véletlenítés

Ahelyett, hogy a folyamatok determinisztikusságát követelnénk meg, néha hasznos, ha a folyamatoknak lehetővé tesszük a véletlenszerű választást, ami bizonyos adott valószínűségi eloszlásokon alapul. Mivel az alap szinkron rendszer modellje ezt nem teszi lehetővé, kiegészítjük a modellt, az üzenetgeneráló és átmeneti függvényekhez hozzáveszünk egy új *véletlen függvényt*, ami a véletlenszerű választás lépéseit valósítja meg. Formálisan, az automata leírását a minden P_i csúcshoz megadott *véletlen*$_i$ összetevővel bővítjük; *véletlen*$_i(s)$ minden s állapotra egy, az *állapotok*$_i$ egy részhalmazán értelmezett valószínűségi eloszlást ad. A végrehajtási sorozat minden menetében először a *véletlen*$_i$ véletlen függvényt alkalmazzuk. Ez meghatározza az új állapotokat, és ezután az *üzenetek*$_i$ és *átmenetek*$_i$ függvényeket a szokásos módon használhatjuk.

A véletlenszerű algoritmusra a végrehajtási sorozat formális leírása nem csak az állapot-hozzárendeléseket és üzenet-hozzárendeléseket, hanem a véletlen függvényekre vonatkozó információt is tartalmazza. A rendszer egy *végrehajtási sorozata* egy végtelen

$$C_0, D_1, M_1, N_1, C_1, D_2, M_2, N_2, C_2, \ldots$$

sorozat, ahol mindegyik C_r és D_r egy állapot-hozzárendelés és mindegyik M_r és N_r egy üzenet-hozzárendelés. D_r ábrázolja az r-edik menet utáni véletlen választással kapott új folyamatállapotokat.

Azok az állítások, melyeket a véletlenszerű rendszerekkel számítunk ki, rendszerint valószínűségeket tartalmaznak és azt állapítják meg, hogy bizonyos eredmények legalább mekkora valószínűséggel érhetők el. Amikor egy ilyen állítást megfogalmazunk, a cél általában az, hogy igaz legyen minden bemenetre, és a hibákat is tartalmazó rendszerek esetében fennálljon bármilyen hiba esetén is. A bemeneteket és a hibaeseteket modellezve rendszerint feltételezünk egy fiktív, ***ellenfélnek*** nevezett elemet is, ez szabályozza a bemenetek és a hibaelőfordulások véletlenszerűségét, és egy valószínűségi állítás azt mondja, hogy a rendszer helyesen viselkedik minden megengedett ellenféllel versenyezve. Ennek a kérdésnek az általános vizsgálata kívül esik e könyv témakörén, csak azokkal a speciális esetekkel foglalkozunk, amelyek szükségesek lesznek.

2.8.. Megjegyzések a fejezethez

Az állapotokkal rendelkező gépek modelljének általános fogalma a hagyományos véges állapotú automata modellekből származik. A véges állapotú gépekre vonatkozó alapvető tananyagok sok egyetemi tankönyvben megtalálhatók, ilyen például Lewis és Papadimitriou [195] és Martin [221] könyve. Az itt leírt állapotokkal rendelkező gépmodell speciális esetei az osztott számítások elméletének számos cikkéből származtathatók, például Fischer és Lynch [119] bizánci megegyezésre vonatkozó cikkéből.

Az invariáns állítások gondolatát először Floyd [124] javasolta soros feldolgozású programokra, majd Ashcroft [15] és Lamport [175] általánosította párhuzamos programokra. Hasonló gondolat számos helyen található. A szimuláció gondolatának is sok forrása van. Az egyik legfontosabb a soros feldolgozású programokban, például Liskov CLU programnyelvében [198], Milner [228] és Hoare [158] munkáiban megvalósított adatabsztrakciókra vonatkozó korai munka. Ilyen későbbi munkák, amelyek kiterjesztették a fogalmat a párhuzamos programokra, Park [236], Lamport [177], Lynch [203], Lynch és Tuttle [218] és Jonsson [165] cikkei.

Tárgymutató