tanulmányok 138/1982

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

SZÁMÍTÓGÉP-HÁLÓZATI PROTOKOLLOK FORMÁLIS SPECIFIKÁLÁSA ÉS VERIFIKÁLÁSA

Irta:

KOVÁCS LÁSZLÓ

Tanulmányok 138/1982

A kiadásért felelős:

VÁMOS TIBOR

Főosztályvezető:

Bakonyi Péter

Osztályvezető:

Csaba László

ISBN 963 311 147 1

ISSN 0324-2951

TARTALOMJEGYZÉK

Oldal

Oldal

Mottó:

"Ha az emberek szabatosan meghatároznák azokat a szavakat, amelyeket használnak, kevesebb vita volna."

Voltaire

1. BEVEZETÉS

1.1 Előszó

Napjainkban a számitógép-hálózatok felhasználása iránti igény jelentős mértékben megnövekedett. A társadalmi élet különböző területein (gazdaság, politika, kultura, tudomány) jelentkeznek ezek az igények. A számitógép-hálózatok ilyen gyors ütemü elterjedése felveti a hálózatok tervezésének és megvalósitásának müszaki és tudományos problémáit. Jelen dolgozat egy ilyen probléma csoporttal, a számitógép-hálózati kommunikációs protokollok tervezésének különböző kérdéseivel foglalkozik.

A számitógép-hálózatok olyan elosztott rendszerek, amelyekben független számitógépek és számitógép rendszerek egymással információt, adatokat cserélnek és erőforrásaikat megosztják. A számitógép-hálózatokban az információ cserét adatátviteli eljárások (protokollok) rendszere biztositja.

A számitógép-hálózat hatékony és megbizható müködése nagy mértékben függ a hálózat belső kommunikációjának megszervezésétől, az alkalmazott protokoll rendszertől. A jelenleg elterjedt protokollok, protokoll rendszerek bonyolultságukat tekintve már-már meghaladják az emberi áttekintőképesség határát. Manapság még nincsenek meg azok a szilárd elméleti eredmények, amelyek alapját képezhetnék a protokollok szisztematikus tervezésének. Hiányzik a protokollok formális kezelésének egységes elmélete, egy olyan szemléletmód, metodológia, amely segitségével el lehetne dönteni a protokollok helyességét, össze lehetne hasonlitani a különböző protokollokat a megbizhatóság, hatékonyság stb. szempontjából, kiküszöbölve a tervező személyétől, tapasztalatától függő szubjektiv tényezőket. Az intenziv protokoll tesztelési eljárások - bár nagy számu hiba felderitését teszik lehetővé - nem garantálják a protokoll helyes müködését. Ennek következményeképpen nincs biztositék arra, hogy üzemszerü körül-

mények között a protokoll mindig az elvárt módon viselkedik. A számitógép-hálózatok széles körü elterjedésével párhuzamosan megnövekszik annak a veszélye, hogy a rejtett, hibás protokoll müködés esetleg felderitetlen marad, aminek beláthatatlan következményei lehetnek, különösen az olyan hálózatok esetén, amelyek létfontosságu feladatokat látnak el (védelmi rendszerek). Mindenképpen indokolt tehát olyan formális specifikációs eljárás kidolgozása, amely azon tul, hogy segitségével egyértelmüen, pontosan leirhatjuk az adott protokoll müködését, lehetőséget teremt valamilyen formális helyesség ellenőrző módszer alkalmazására még az implementációt megelőző fázisban. Ehhez ugyanis fontos gazdasági érdek is füződik. Amennyiben egy adott protokoll hibás volta, vagy az adott körülmények közötti alkalmatlansága csak az implementáció után derül ki, akkor a protokoll kijavitása, megváltoztatása esetleg az egész implementáció áttervezését követelheti meg, amelynek hatalmas anyagi konzekvenciái lehetnek. Ennek a megelőzésére az utóbbi években a számitógép-hálózati protokollok specifikálásával és verifikálásával kapcsolatos kutatások intenzivebbé váltak. A kutatók számos, egymástól sok esetben nagymértékben különböző eljárást dolgoztak ki a protokollok formális leirására, protokoll modellezésre, és megtették az első lépéseket a protokoll verifikáció irányába is. Hiányzik azonban a különféle módszerek egységes kritérium rendszer alapján történő összehasonlitása, a módszerek közötti kapcsolatok, azonosságok és különbségek feltárása. Hiányzik egy olyan egységes elméleti rendszer, amely keretében a különféle módszerek elhelyezhetők lennének és megszünnének az alapfogalmak értelmezésében jelenleg tapasztalható jelentős eltérések is. Az elméleti letisztulás alapvető gátja a diszciplina igen gyors fejlődése, amely azonban legtöbbször a technológia fejlődésére és nem pedig az uj elméleti eredményekre vezethető vissza. Az egységes protokoll elmélet hiánya egyre inkább a továbblépés akadályának tekinthető.

Jelen munkában a protokoll tervezés egy egységes metodológiájára teszünk javaslatot, amely egy protokoll specifi-

kációs nyelvre épülő protokoll definiálási módszert és a hozzá kapcsolódó számitógéppel támogatott verifikálási eljárást tartalmaz. A következőkben a dolgozat felépitéséről szólunk.

A csomagkapcsolt számitógép-hálózatok referencia modelljének rövid bemutatása után a protokoll specifikációs és verifikációs módszerekről adunk áttekintést, nagyobb figyelmet szentelve a magyar nyelven még nem ismertetett eljárásoknak. A módszereket az állapotátmeneten, a programozási nyelveken alapuló, illetve az egyéb módszerek csoportjába soroltuk.

A 3. fejezetben részletesen tárgyaljuk a formális protokoll specifikációval és verifikációval szemben támasztott követelményeket. Elemezzük a protokoll specifikációs nyelv elvárt tulajdonságait a nyelv által betöltött szerepből kiindulva.

A 4. fejezet a protokoll tervezés egységes módszertanát mutatja be.

A következő fejezetben fokozatosan felépitjük a számitógép-hálózati protokollok egy absztrakt matematikai modelljét, amely a később bemutatott specifikációs nyelv alapját képezi. A matematikai modell lehetőséget teremt a protokollok bizonyos tulajdonságainak formális értelmezésére is.

Az 5. fejezet a protokoll leirásra kialakitott specifikációs nyelv elemeit mutatja be. A követelményekből kiindulva, a MODULA-2 programozási nyelv alapvető konstrukcióit felhasználva, szem előtt tartva az előző fejezet tartalmazta matematikai modellt, speciális uj nyelvi elemek felhasználását javasoljuk. A nyelv alkalmazásának illusztrációjaként a Bitalternáló protokoll specifikációját közöljük.

A következőkben a protokoll tulajdonságok formális megfogalmazása után az állapotelérhetőségi analizist mutatjuk be. Részletesen tárgyaljuk azt, hogy az egyes protokoll tulajdonságok felderitésében milyen szerepet kaphat a fent emlitett verifikációs módszer. Kimutatjuk, hogy az állapote-

lérhetőségi analizis bővitésével a protokollok funkcionális helyességének ellenőrzése is lehetővé válik.

A 9. fejezet a bővitett elérhetőségi analizisen alapuló, számitógéppel támogatott interaktiv protokoll verifikáló rendszer komponenseit tartalmazza. Az egyes megvalósitott modulok leirása után a verifikáló rendszer müködésével kapcsolatos első tapasztalatokról számolunk be.

Az összefoglalás után a továbblépés lehetőségeiről szólunk néhány nyitott probléma felsorolásával. Az értekezést irodalomjegyzék zárja.

1.2 Köszönetnyilvánitás

Szeretnék ezuton is köszönetet mondani dr. Harangozó Józsefnek fáradhatatlan támogatásáért, tanácsaiért, amelylyel munkámat nagymértékben elősegitette.

Köszönetet mondok az MTA-SZTAKI Hálózat és Távadatfeldolgozási Osztály munkatársainak, közülük is elsődlegesen Böszörményi Lászlónak, akitől igen sok értékes segitséget kaptam a programok MODULA-2 nyelvü megvalósitásakor.

Köszönetemet fejezem ki dr. Bach Ivánnak, akinek érdekes szemináriumain elsajátitott gondolkozásmód jelentős segitséget nyujtott e tématerület feldolgozásakor.

Köszönöm a "Protokoll együttmüködés" tagjainak és személy szerint dr. Tarnay Katalinnak a támogatást és azt a figyelmet, amellyel munkámat végigkisérték.

Utoljára de nem utolsó sorban köszönet illeti a "Korszerü folyamatirányitó rendszerek tervezése" szakszeminárium minden résztvevőjét, azért az aktiv vitaszellemért, amely szemléletmódom csiszolódásához vezetett.

Budapest, 1982. augusztus 15.

Kovács László

1.3 Jelölések

A matematikában és a programozás elméletben megszokott jelöléseket használjuk. A 2. fejezet szakirodalmi áttekintését kivéve a jelölés rendszer konzisztens. A 2. fejezetben a különféle irodalmi hivatkozások bemutatásánál az eredeti közlemények jelölés rendszerét tartjuk meg, amelyet mindenhol részletesen megmagyarázunk. Ebből következőleg természetes módon előfordulhat az, hogy ugyanannak a dolognak más-más jelölés felel meg vagy viszont.

A dolgozatban néhány helyen a témakör modern voltából következőleg a nehézkes, nem létező vagy nem elterjedt magyar terminológia helyett a megfelelő angol kifejezéseket használjuk. A bemutatott programok, program részletek MODULA-2 nyelven [Wirth 80] iródtak. A programok azonositói a programozási nyelvek kulcsszavainak angol volta miatt ugyancsak angol eredetüek.

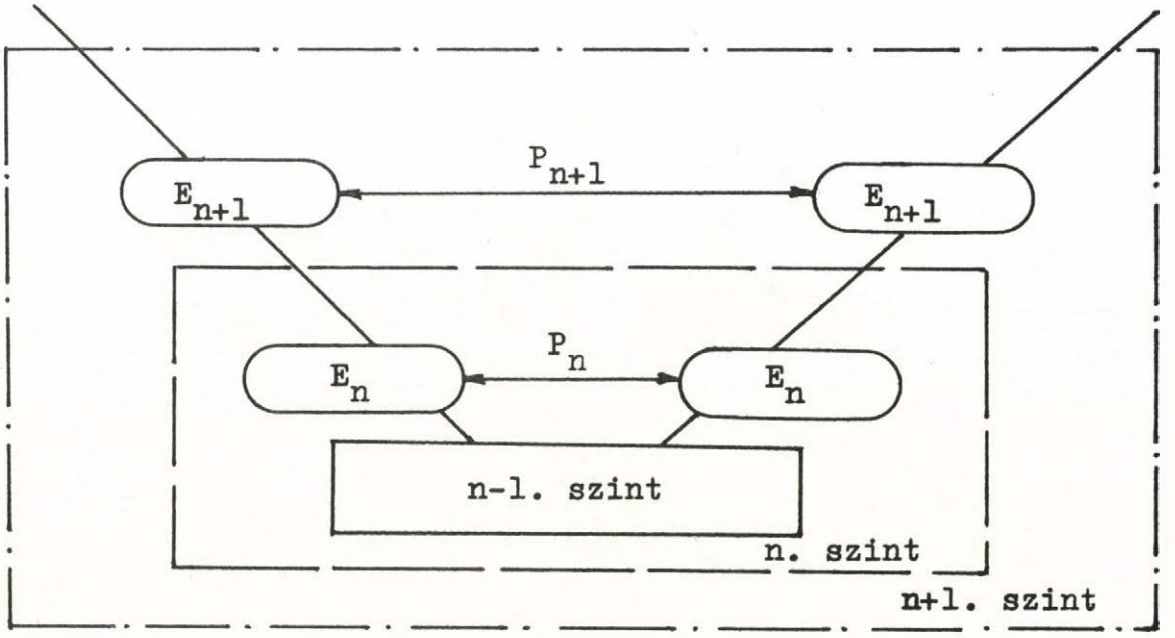
1.4 Csomagkapcsolt számitógép-hálózat referencia modellje

A számitógép-hálózat kommunikációjának alapja a protokoll. A protokoll olyan, a számitástechnikai elemek (hostok, terminálok, processek stb.) közötti információ cserére vonatkozó kölcsönösen elfogadott megállapodások összessége, amelyek biztositják a megbizható, gyors kommunikációt. A számitástechnikai elemek fogalmából absztrakció utján eljuthatunk az entitás fogalmához, amely tágabb értelemben egy tulajdonságaival jellemzett objektum, mig szükebb értelemben a környezetével információ cserére képes dolgok általános neve. Az entitás fogalmának felhasználásával tehát a protokoll két vagy több entitás között értelmezett kommunikációs célu szabály gyüjtemény. A következőkben ezen protokoll és entitás értelmezés alapján bemutatjuk a számitógép-hálózatok un. referencia modelljét. A protokoll illetve entitás pontos formális definicióját az 5. fejezetben fogjuk megadni.

A számitógép-hálózatok elosztott rendszereknek tekinthetők. Az elosztott architekturák létesitése többek között a réteges (hierarchikus), kaszkád, csillag és háló struktura kialakitási módszereken alapul [Pouzin és m. 78]. A fenti struktura kialakitási módszerek a jelenlegi technikai, technológiai szint következményei, igy az eddigiektől eltérő technológiák esetleg uj elveken felépülő architekturák megjelenését vonhatják maguk után.

Hierarchikus (réteges) szervezés

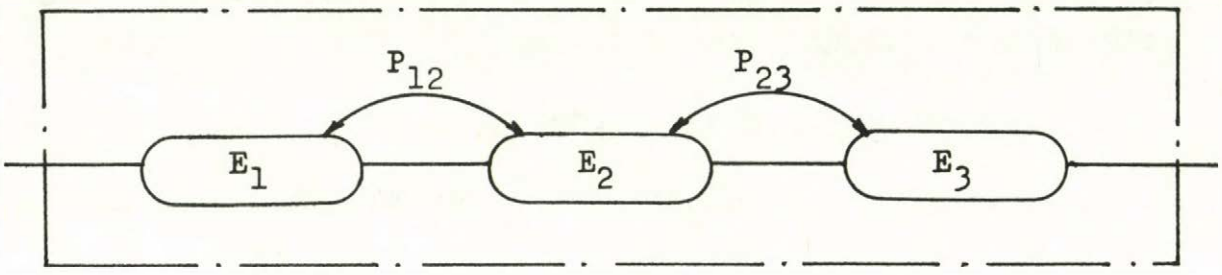
A számitógép-hálózatok területén manapság a leggyakrabban alkalmazott struktura a réteges felépités. Kialakitásának alapja az, hogy két valamilyen protokoll szerint kommunikáló entitás felhasználható arra, hogy más entitások számára kommunikációs szolgáltatást nyujtson. Ilyen módon épül fel az 1.1 ábrán látható hierarchikus rendszer. A rendszer azonos szintjein lévő entitások a köztük értelmezett protokollnak megfelelően cserélnek információt ugy, hogy az alacsonyabb szint kommunikációs szolgáltatásait veszik igénybe.



1.1 ábra Hierarchikus szervezés

Kaszkád szervezés

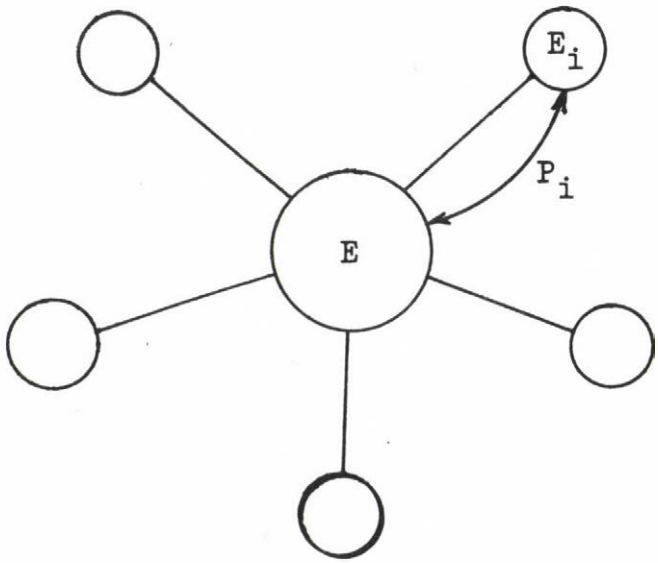
Kaszkád felépitès esetén az entitások sorba rendezettek ugy, hogy minden entitás csak a megelőző és az utána következő entitással kommunikálhat valamilyen protokoll szerint.



1.2 ábra Kaszkád struktura

Csillag struktura

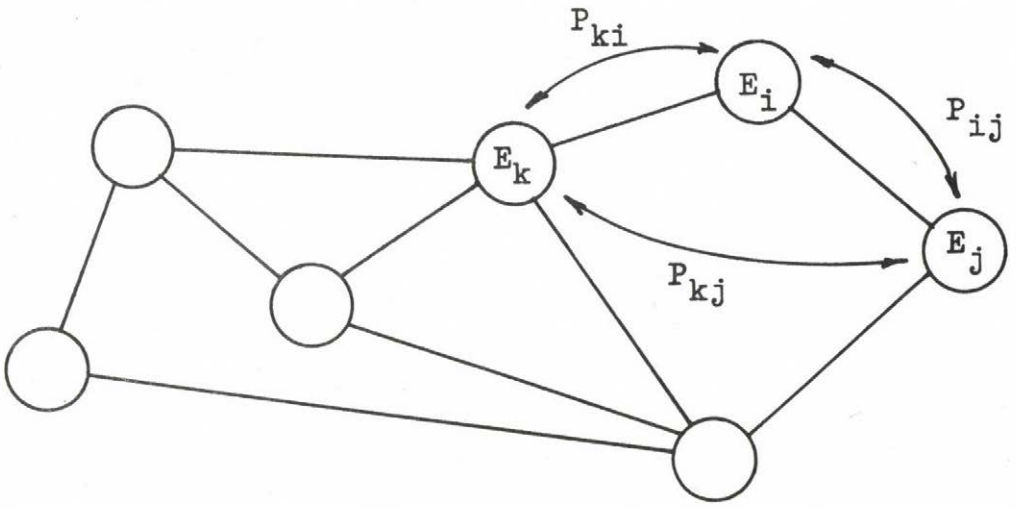
Történetileg a legkorábban kialakult struktura a csillag felépitès, melyben az entitások csak a központi entitással illetve azon keresztül kommunikálhatnak.



1.3 ábra Csillag struktura

Háló struktura

A háló strukturában bármely két entitás kommunikálhat egymással valamilyen protokoll szerint.

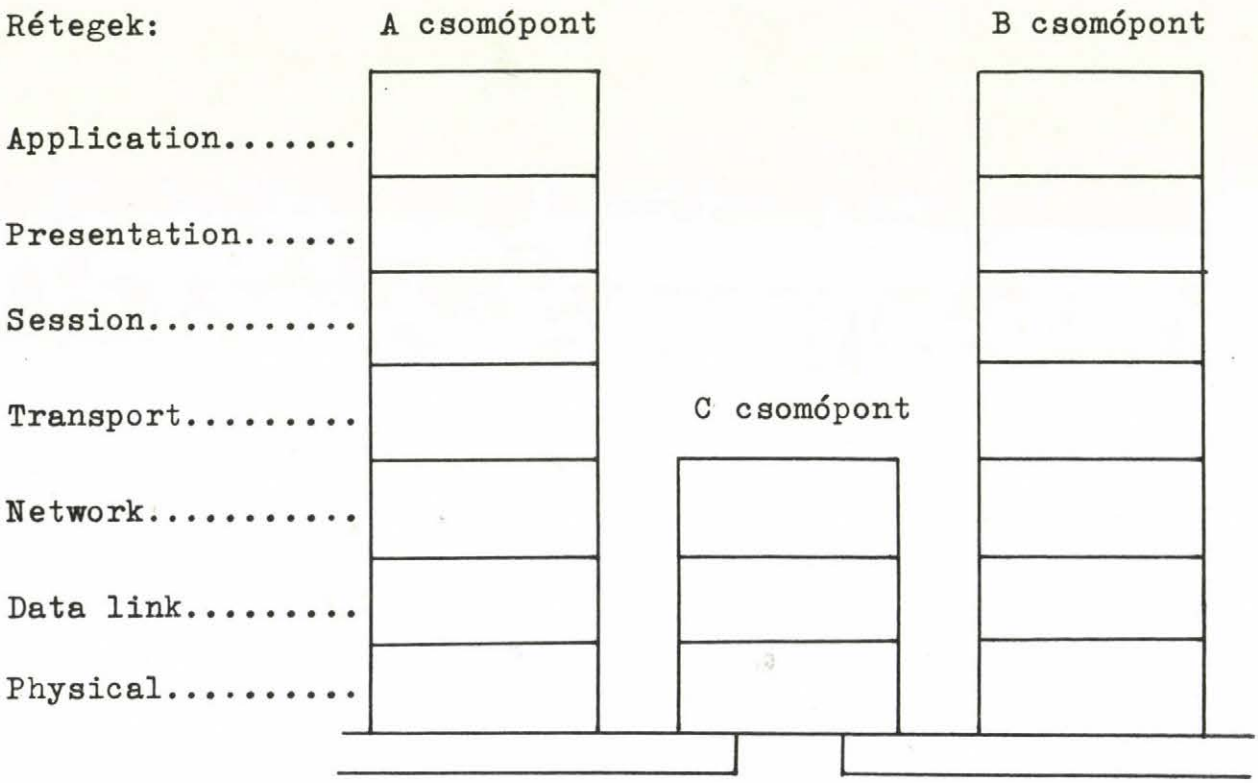


1.4 ábra Háló struktura

A számitógép-hálózatok réteges felépitésüek. A rétegek közötti kapcsolatot az interfészek biztositják. Az azonos rétegekben elhelyezkedő entitások közötti kölcsönhatás szabályait pedig protokollok irják elő. Amennyiben rögzitjük az interfészek és protokollok szabályait, akkor az eltérő tipusu és belső müködésü rendszereket is hálózatba foglalhatjuk, feltételezve persze, hogy megvalósitják az interfész és protokoll szabványokat.

A referencia architektura (referencia modell) fogalmát azért vezették be, hogy egyszerübb legyen az elosztott rendszer interfészeire és protokolljaira hivatkozni. A korábban kialakitott rendszerek ugyanis az architekturális különbségeken tul más-más terminológiát alkalmaztak, amely jelentősen megnehezitette a különböző funkcionális részekre történő hivatkozást. A számitógép-hálózatok egységes referencia architekturájának kialakitása jelenleg is folytatódó nemzetközi szabványositási munka. Az ISO-OSI [ISO 81a] szabványtervezet

definiálja a Nyilt Rendszer Architekturát, amely egy 7 szintes hierarchikus rendszer (1.5 ábra). Azokat a rendszereket amelyeknek logikai képe megfelel e 7 szintes modellnek Nyilt rendszernek nevezi. A szabványtervezet az ilyen Nyilt Rendszerek egymás közötti kommunikációjával foglalkozik. A következőkben áttekintjük az egyes rétegek funkcióit, de részletes ismertetés helyett az irodalomra utalunk.



1.5 ábra Az ISO Nyilt Rendszer referencia architekturája

Alkalmazói réteg (Application Layer)

A Nyilt Rendszer Architektura legfelső rétege. A réteg entitásai között értelmezett protokollok a hálózat felhasználójának elosztott számitástechnikai szolgáltatásokat nyujtanak. Ilyen szolgáltatások lehetnek például a

- kommunikációs partnerek azonositása
- szolgáltatások minőségének megállapitása
- szinkronizálás a kooperáló partnerek között stb.

A réteg lehetőséget teremt a rendszer müködésének szabályozására (system management) is.

Adatábrázolási réteg (Presentation Layer)

A réteg adat transzformációs (karakterkód és készlet konvertálás), adat formázási szolgáltatásokat nyujt az alkalmazói réteg entitásainak. Lehetőséget teremt a transzformációs szolgáltatások, adat formátumok kiválasztásának módositására is.

Kapcsolat felépitési réteg (Session Layer)

E réteg az adatábrázolási réteg kooperáló entitásai közötti kapcsolatok (session-connection), adatátviteli utak létrehozását, fenntartását és felbontását végzi. Feladata még ezen tul az adatátvitel szabályozása, szinkronizálása.

Transzport réteg (Transport Layer)

A transzport réteg egységes, transzparens adatátviteli szolgáltatást nyujt az alatta lévő rétegek támogatásával, de azok tulajdonságaitól függetlenül. Ellenőrzi a forrás és cél állomás közötti több csomóponton keresztüli adatforgalmat.

Hálózati réteg (Network Layer)

A transzport réteg entitásai üzeneteinek irányitását (pl. utképzés), két szomszédos csomópont közötti adatfolyam vezérlést, hibaellenőrzést végez.

Adatkapcsolati réteg (Data Link Layer)

Két vagy több szomszédos csomópont közötti adatkapcsolat létesitésének, fenntartásának és bontásának szabályait irja elő, biztositva a hibamentes adatáramlást.

Fizikai réteg (Physical Layer)

A réteg protokolljai előirják a kapcsolatok fizikai és

elektromos jellemzőit.

Az elkövetkező fejezetekben számitógép-hálózati rendszerek alatt a Nyilt Rendszer Architekturával jellemezhető rendszereket értünk. A 6. fejezetben ismertetett protokoll specifikációs nyelv jól alkalmazkodik a referencia modell alapvető strukturájához, a réteges felépitéshez.

2. IRODALMI ÁTTEKINTÉS

A fejezetben az számitógép-hálózati kommunikációs protokollok specifikálásával és verifikálásával foglalkozó főbb szakirodalmi közleményeket tekintjük át. A magyar nyelvü disszertációkban [Harangozó 78b],[Margitics 81] részletesen elemzett módszereket csak vázlatosan mutatjuk be, figyelmünket inkább a fenti értekezésekben nem szereplő eljárások felé forditjuk. Terjedelmi okokból ugyancsak nem tudunk foglalkozni azokkal a közleményekkel, amelyek bár nem specifikációs és verifikációs módszereket tartalmaznak, de mégis jelentősen hozzájárultak e munka elkészitéséhez.

A legtermészetesebb protokoll definiálási "módszer" az élő nyelven történő protokoll leirás. A hivatalos protokoll definiciók legtöbbje még manapság is a természetes élő nyelvi leirást használja. A természetes nyelv legnagyobb előnye a könnyü használatban és megérthetőségben rejlik. Közismert azonban, hogy ez esetben nehéz kiküszöbölni a szöveg többértelmüségeit, nehéz megállapitani azt, hogy a szöveg foglalkozik-e elegendő részletességgel a protokoll különféle részeivel vagy sem. Ilyen leirási módszer esetén a fomális protokoll verifikáció csaknem lehetetlenné válik. A protokollról un. szövegelemzéssel kapott információk mennyisége és minősége ugyanis nem áll arányban a befektetett munkával és legtöbbször végeredményként kiderül az, hogy a természetes nyelven leirt protokoll "definició" nem teljes, sőt inkonzisztens.

Ezeken a problémákon (már viszonylag korán) ugy próbáltak segiteni, hogy a dokumentumok formáját (külalakját) szabályozták. Kialakultak az un. kötött szintaxissal rendelkező leirási módszerek, amelyekben a dokumentumok felépitésére vonatkozó előirásokon tul erőteljesen korlátozták az alkalmazható nyelvtani szerkezetek, szavak (pl. kötőszavak) számát is. Ezek az ajánlások szemantikai megkötést

még nem tettek. A következő lépés az un. formáknak alávetett specifikációs módszerek megjelenése volt. Ezek közül legismertebb a különféle tipusu folyamatábrákat felhasználó leirási módszer csoport. A folyamatábrával történő leirás mindaddig nem jelentett előrelépést a kötött szintaxissal rendelkező leirásokhoz képest, ameddig meg nem jelentek a folyamatábra blokkjain belül használt mesterséges (formális) nyelvi elemek. Ma már a folyamatábrát az áttekintést elősegitő grafikus segédeszköznek tekintjük csupán, amely megkönnyitheti a rövidebb szekvenciális programok vezérlésének végigkövetését, mig a párhuzamos programok folyamatábrákkal történő illusztrálása általában reménytelen feladat.

Az ismertetett protokoll specifikációs és verifikációs módszereket három fő csoportba, az állapotátmeneten alapuló, a programozási nyelven alapuló és az ezektől eltérő módszerek csoportjába soroltuk. Az irodalomban hibrid módszerek néven nevezettek egy részét az állapotátmeneten alapuló, másik részét (specifikációs nyelvek) a programozási nyelveken alapuló módszerek között tárgyaljuk.

2.1 Állapotátmeneten alapuló módszerek

2.1.1 Véges automaták

A véges automatát alkalmazó eljárások alkotják a legkorábban kialakult és mind a mai napig legnépszerübb protokoll specifikációs módszereket. Elterjedtségüket egyszerüségüknek, könnyü megérthetőségüknek köszönhetik. A véges automata matematikai szempontból egy halmaz hetes:

$$(Q, \Sigma, \Delta, \delta, \mu, q_o, q_v)$$

ahol a

- Q az állapotok véges halmaza
- Σ input alfabeta
- Δ output alfabeta
- δ állapotátmenet függvény
- μ kimenet függvény
- q_o kezdő állapot
- q_v végállapot(ok)

és mind a hét halmaz véges. A Σ és Δ halmazok azokat a "jeleket" tartalmazzák, amelyeket az automata a külvilág felöl kap illetve amelyeket a külvilág felé ad. A δ kétváltozós állapotátmenet függvény leképezi az állapothalmaz és az input jelek halmazának elemeiből álló párokat az állapothalmaz elemeire ($\delta : Q \times \Sigma \longrightarrow Q$). A μ kétváltozós kimenet függvény pedig a bemeneti jel és az aktuális állapot segitségével előállitja a kimeneti jelet ($\mu : Q \times \Sigma \longrightarrow \Delta$). Összefoglalva tehát az automata müködését: kezdetben az automata a $q_o \in Q$ kezdő állapotot állitja be, majd a beérkező input jel hatására állapotváltozás közben output jelet állit elő és ez mindaddig igy folytatódik, ameddig az automata nem ér valamely $q_v \in Q$ végállapotba, mely esetén a müködés abbamarad. A protokoll leirás szempontjából célszerü az a feltételezés, hogy un. inicializálással az automata bármely pillanatban a kezdő állapotba vihető. Ez a feltételezés egyet jelent azzal, hogy a Σ -ban van egy kitüntetett elem

a σ_{init} és a $Q \times \sigma_{init} \longrightarrow q_o$, $Q \times \sigma_{init} \longrightarrow \emptyset$ leképezések definiáltak (inicializálás közben az automata nem ad ki output jelet).

A véges automaták a protokollok természetes absztrakt modelljének tekinthetők, a protokoll entitások tevékenységei, tevékenység sorozatai ugyanis külső események hatására generálódnak. Ilyen külső esemény lehet például valamilyen üzenet megérkezése, a felsőbb protokoll szint parancsa, esetleg belső időhatár tullépés (timeout) jelzése. A valóság ilyen eseményeiből absztrakcióval kaphatjuk meg a Σ és Δ input és output jelek véges halmazát. A protokollok véges automatával történő leirása a protokoll entitásoknak megfelelő véges automaták (legtöbbször két automata) definiálásából áll, bár a protokoll müködését egyetlen egy automata is jellemezheti. Utóbbi esetben bizonyos állapotszám elérésekor természetes módon felvetődik az automata dekompoziciójának szükségessége, igy az előző megoldást (entitásonként egy-egy automata) adekvátabbnak tekinthetjük, amely jobban tükrözi az esetleg távoli entitások közötti kommunikáció lényeges vonásait. Az automata rendszerrel történő leirásnál meg kell oldani az automaták összekapcsolásának problémáját is.

Ezen alapgondolatok alapján nagy számu protokoll specifikációs módszert dolgoztak ki, amelyek egymástól legtöbbször csak kevéssé térnek el.

A [Bochmann 78] közleményben a szerző a protokoll rendszer kommunikációs komponenseinek egy-egy véges automatát feleltet meg, amelynek állapotátmeneteit elnevezi (tipusokba sorolja). Az i-edik komponens s_i állapotából például az s_i' állapotba a t tipusu állapotátmenet hatására kerül.

$$s_i \xrightarrow{t} s_i'$$

Bevezeti a komponensek közötti direkt csatolás (direct coupling) fogalmát, amely nem más, mint a komponenseket jellemző

véges automaták közötti kommunikáció (kapcsolatok) egy mechanizmusa. Két komponens (i,j) közötti direkt csatolást un. direkt csatolt állapotátmenetek halmazával adja meg $\{t_i \| t_j\}$, ahol t_i az i-edik komponens, mig t_j a j-edik komponens valamilyen állapotátmenete. A direkt csatolt állapotátmenetek csak párhuzamosan (szinkronban) hajthatók végre. A direkt csatolt állapotátmenetek realizálásának egyik módja az, ha az egyik automata output jele megegyezik a másik automata megfelelő input jelével. Ez esetben az automaták közötti kommunikáció pufferelés nélkül megy végbe, hasonlóan a [Hoare 78] nyelvi javaslathoz. A kommunikációs protokollok modellezéséhez a réteges protokoll strukturát (1.4 fejezet) tételezi fel. A vizsgált protokoll réteg alatti rétegek tulajdonságait a medium komponens (entitás) segitségével modellezi, amelynek leirására ugyanugy mint a többi komponens esetén direkt csatolt véges automatát használ. A dolgozat foglalkozik a protokoll validáció különböző kérdéseivel is. A validáció alapjának az elérhetőségi analizist tartja, mely segitségével ki lehet mutatni a deadlock szituációkat, a protokoll ciklusokat. Vázlatosan értelmezi a liveness, az önszinkronizáció és stabilitás fogalmakat. Felveti a véges állapotu gépekkel történő megközelités alapproblémáját az un. "állapotrobbanás" jelenségét, amely leggyakrabban abból adódik, hogy például a sorszámozás miatt jelentősen megnövekszik a komponens állapotainak száma, amely megneheziti, sőt sok esetben gyakorlatilag lehetetlenné teszi az elérhetőségi analizis manuális végrehajtását. Bevezeti az üres medium (empty medium) absztrakciót, amely jelentősen csökkenti az állapotok számát, ugyanis csak azokat az állapotokat tekinti, amely esetén a medium üres. Ennek az absztrakciónak az alkalmazása különösen az olyan protokolloknál hasznos, amelyeknél egy adott pillanatban csak kis számu üzenet van uton. A vizsgált protokoll réteg globális (a felső szint szemszögéből értelmezett) müködésének leirására szolgáló egyetlen automata konstruálására a következő javaslatot teszi: a felső szinttel direkt csatolásban lévő állapotátmenetek felhasználásával

a protokoll szint globális müködését leiró reguláris kifejezést szerkeszti meg, majd felhasználja a reguláris kifejezések és a véges automaták ekvivalenciáját és előállitja a protokoll réteg felső szint számára nyujtott szolgáltatásait leiró automatát. (A későbbiekben ez a szolgáltatás és protospecifikáció erőteljes szétválásához vezet.) Az elmondottak illusztrálására a Bitalternáló illetve az X.25 Set-Up és Clearing eljárásainak specifikációját és elemzését mutatja be.

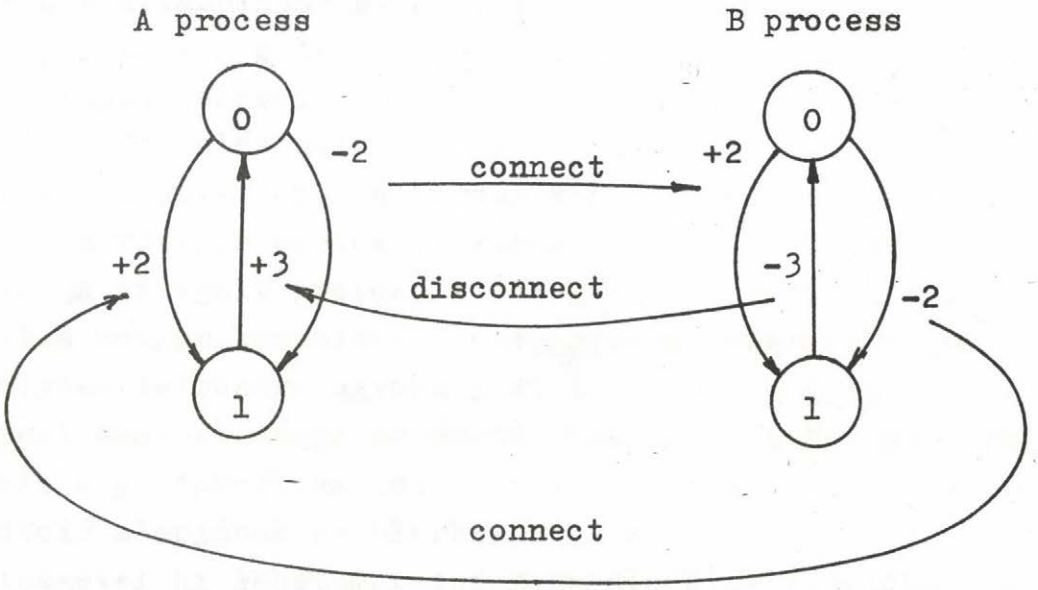
A direkt csatolt véges automaták hierarchikus rendszerével történő protokoll leirás szép példája az [ASZH 81], amely az Akadémiai Számitógép Hálózat referencia modelljét tartalmazza. Az automaták input és output jelei (primitivek) paraméterekkel kiegészitettek.

Mindkét előbb emlitett közlemény a véges automaták megadására az állapotdiagramot használta, amely a véges automata grafikus reprezentánsa. Kis állapotszámig az állapotdiagram megkönnyiti az áttekintést, de nagyobb állapot és állapotátmenet szám esetén az ábra menthetetlenül zavaros, áttekinthetetlen lesz.

2.1.2 A "duologue" és a "perturbációs" módszer

Az IBM zürichi laboratóriumában kutatásokat folytattak a protokoll validáció automatizálásának irányában. Az eredményeket 1977 óta megjelenő cikkek formájában tették közzé. Zafiropuló kidolgozta a protokoll verifikáció "duologue" elméletét, amely az ugyancsak általa kidolgozott formális protokoll leirási módszerre épül. West módositotta a "duologue" elméletet, kiküszöbölte az eredeti megfogalmazás néhány hiányosságát és a "fázis diagram" fogalmának bevezetésével, az "állapot perturbációs" protokoll verifikáló módszer kidolgozásával megteremtette a protokoll validáció automatikus végrehajtásának lehetőségét. A következőkben ezeket a módszereket mutatjuk be.

Zafiropuló [Zafiropuló 78] a protokollokat irányitott gráf párokkal irja le. Az irányitott gráf lényegében a protokoll entitásokat véges automatával leiró módszer állapotgráf reprezentánsával ekvivalens.



2.1 ábra Protokoll reprezentáció irányitott gráffal

A 2.1 ábrán egy egyszerü példán illusztráljuk a módszert. A gráf párok irányitott éleit az állapotátmenetek során küldött és fogadott eseményeket jelző előjeles számok cimkézik. A pozitiv előjel az esemény fogadását, a negativ pedig a küldését jelenti. Kezdetben mindkét process entitás a "0"-val jelölt (disconnect) állapotban van. A kapcsolat felvételét bármelyik process a 2-es esemény elküldésével kezdeményezheti. A kapcsolat felvételekor az entitások az "1" állapotba kerülnek. A bontást csak a B process indithatja el. Ezen megközelités és korábban bemutatott véges automatákat alkalmazó megközelités közötti látszólagos különbség abban rejlik, hogy esetünkben egy állapotátmenet során csak egyetlen egy esemény küldése vagy fogadása történhet meg, mig a véges automatánál minden állapotátmenetet külső esemény idéz elő. A két leirás közötti szigoru ekvi-

valencia belátható, ha a véges automata Σ input és Δ output jeleinek halmazában az ε (epszilon) üres jel meglétét feltételezzük. A szerző bevezeti a "¯"-sal jelölt speciális esemény (nonevent) fogalmát, amivel harmadik processtől származó interakciókat modellez. A rendszer müködés leirására az irányitott gráfokban értelmezett ut párokat (unilogues) használja. Az ut (unilogue) az adott process kezdő állapotából kiinduló és ugyanoda visszatérő irányitott él sorozat, amelyet az érintett élek cimke sorozatával jellemez. Például a 2.1 ábra rendszerének utjait az

$$A_1 = (-2, +3) \qquad A_2 = (+2, +3)$$
$$B_1 = (+2, -3) \qquad B_2 = (-2, -3)$$

kifejezések adják meg. Az A és B process összes utjait az S_A és S_B halmazok tartalmazzák, amelyek meghatározására a szerző az állapotátmenet mátrix szorzáson alapuló algoritmust dolgozott ki.

$$S_A = \{A_1, A_2\} \qquad S_B = \{B_1, B_2\}$$

A "duologue" két unilogue párositásából áll, ahol az egyik unilogue az egyik, a másik unilogue a másik process egy unilogue-ja. ($[A_i, B_j]$) Az összes lehetséges duologue-t az $S_A \times S_B$ halmaz adja meg. Példaképpen a fenti esetben

$$S_A \times S_B = \begin{bmatrix} [A_1, B_1] & [A_1, B_2] \\ [A_2, B_1] & [A_2, B_2] \end{bmatrix}$$

a duologue mátrix. A mátrixra (annak minden duologue elemére) alkalmazva a VAL függvényt megállapitható a protokoll helytelen viselkedése. A VAL függvény +1 értékü, ha a duologue helyes, Ø értékü, ha a duologue nem fordulhat elő és -1, ha hibás. A duologue helyességét az alábbi feltételek teljesülésétől teszi függővé:

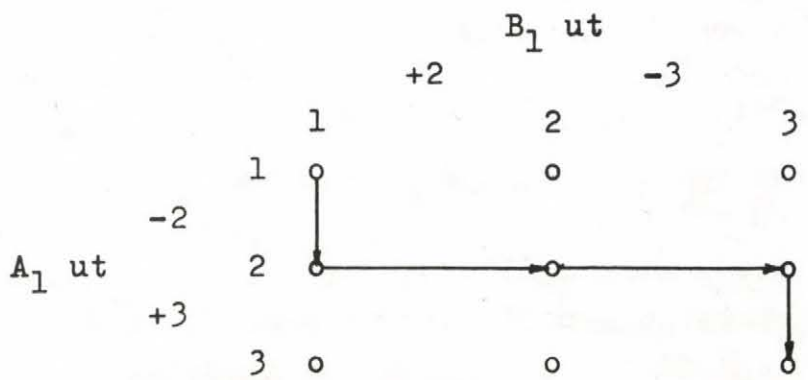
- minden az egyik unilogue által kibocsájtott jelet (eseményt) a másik unilogue-nak fogadni kell,
- minden az egyik unilogue által fogadott eseményt meg kell hogy előzzön az esemény másik unilogue által történő kibocsájtása,
- két ütköző (szimultán) esemény kibocsájtásakor a fogadónak nem szabad az események beérkezési sorrend-

jét előre feltételezni.
Például VAL $[A_1, B_2] = -1$, mert nem teljesül az utolsó feltétel. A VAL függvény konkrét kialakitására vonatkozó elemzést a [Zafiropuló 78] dolgozat szintén tartalmazza.

$$VAL[S_A \times S_B] = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

A protokollok ezen megközelitése az entitások közötti kommunikációs csatornát ideálisnak tételezi fel. Az átviteli hibák modellezése a gráfokban pótlólagos események felvételét követeli meg. Az átviteli hibák pótlólagosan felvett processel történő modellezésére West tett javallatot [Zafiropuló és m. 80].

West "fázis diagram" megközelitésének alapja a duologue elmélet [West 78]. A fázis diagram sorai és oszlopai az egyik illetve a másik process állapotait reprezentálják.



2.2 ábra Az $[A_1, B_1]$ duologue-nak megfelelő fázis diagram

Az állapotátmeneteket függőleges és vizszintes nyilak jelzik, a processeknek megfelelően. Az ábrán a nyilak alapján felderithető az összes lehetséges duologue és elemezhető a korábban megadott szempontok szerint. A fázis diagram módszer számitógépes megvalósitása segitségével az X.21 ajánlás validációját végezték el [Rudin és m. 78], amely körülbelül 2400 duologue analizálását igényelte.

Az "állapot-perturbációs" módszert West alkotta meg [Rudin és m. 78]. Kiküszöbölte a Zafiropuló elmélet alap-

vető hiányosságait. A Zafiropuló féle megközelitésben ugyanis a processeknek minden esetben vissza kellett térniük a kezdeti állapotba és a módszer csak két kommunikáló processt tudott vizsgálni. Az általánositott elmélet véges sok processre terjesztette ki hatókörét és nem követelte meg a processek kezdeti állapotba történő visszajutását (explicit végállapottal rendelkező processek is elemezhetőkké váltak). A módszer a processek leirására a Zafiropuló féle irányitott gráfokat használja. Feltételezi, hogy bármely két process között két egymással ellentétes irányban üzemelő kommunikációs csatorna van, amelyeken keresztül történik a jelek (események) küldése és fogadása. A teljes protokoll rendszer (n processből álló halmaz) globális állapotát egy n x n-es mátrixszal adja meg. A mátrix főátlójában a processek állapotai, mig a többi helyen az adatátviteli csatornák állapotai (tartalma) állnak. A rendszer kezdeti állapotát a komponensek állapotátmenetei megváltoztatják, ugymond "perturbálják". Egy process által kiadott jel először a megfelelő adatátviteli csatornába kerül (megváltoztatva annak állapotát), majd a fogadó process egy állapotátmenete révén "kiveszi" a cstornából a hozzá érkezett jelet. A rendszer globális állapotainak feltérképezésével felderithetők a protokoll ciklusok, deadlock szituációk, meghatározhatók a feltételezett csatorna kapacitások tullépései. Az adatátviteli csatornák nem ideális voltát pótlólagos állaptátmenetek vagy pótlólagos processek bevezetésével modellezi. A szerzők beszámolnak arról, hogy az állapot-perturbációs módszert megvalósitó APL program két nagyságrenddel (?) gyorsabb mint a fázis diagrammos megközelités.

A protokoll validáció algebrai technikát használó duologue mátrix analizis módszere lényegében, mig a perturbációs módszer teljes egészében megegyezik a véges automatáknál emlitett állapotelérhetőségi analizissel, annak ezen formalizmus keretében megvalósuló egy-egy reprezentánsainak

tekinthetők. A módszer ott elmondott előnyei és korlátai ezen módszerek esetén is érvényesek.

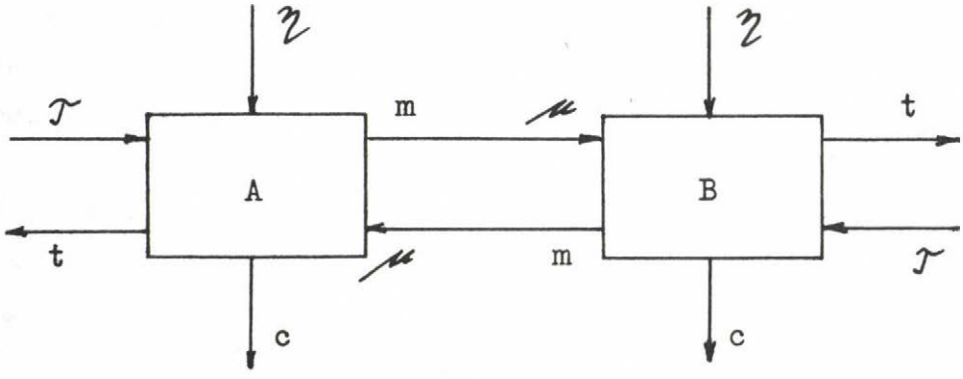
A [Zafiropuló és m. 80] cikk az automatikus, félautomatikus protokoll analizáló és szintetizáló eszközök megvalósitási lehetőségeivel foglalkozik. Beszámol a korábban már emlitett zürichi laboratóriumban folyó kutatások ujabb eredményeiről, arról a szerzők ismerete szerint akkoriban egyedülálló rendszerről, amely számitógépes támogatással logikailag helyes protokollok szintézisét könnyiti meg. A dolgozat első része a protokollok létrehozása közben elkövetett tipikus tervezési hibákat sorolja fel. Értelmezi az állapot deadlock, a "nem specifikált fogadás (unspecified reception), a végrehajthatatlan interakciók, a stabil állapotpár és az "állapot kétértelmüség" (state ambiguities) fogalmakat. Alapvető protokoll analizáló technikaként a perturbációs módszert mutatja be. Egy egyszerü protokoll példáján illusztrálja azt, hogy a perturbációs módszer alkalmas a tipikus tervezési hibák kimutatására. Az állapot deadlock kimutatása egyet jelent az elérhetőségi fán olyan globális állapot megkeresésével, amely esetben a mediumok (adatátviteli csatornák) üresek és a kommunikációs komponensek (processek) nem tudják megváltoztatni állapotukat. A nem specifikált fogadás olyan deadlock helyzet, amelyben a mediumok közül legalább egy nem üres. A deadlock szituációk ilyen osztályozását a deadlock helyzetek megszüntetésének különböző módja indokolhatja. A hiányos specifikáció pótlólagos állapotátmenet segitségével feloldható, mig az állapot deadlock megszüntetése bonyolultabb feladat lehet. Az automatikus verifikációs rendszerrel nyert tapasztalatok vezettek az automatikus protokoll szintézis probléma körének behatóbb vizsgálatához. A szerzők az adott formalizmus (protokoll leirási módszer) keretében olyan szabályok (production rules) kidolgozását kisérelték meg, amelyek betartása a korábban emlitett tervezési hibáktól mentes protokollok létrehozását tesz lehetővé. A szabályok részletes ismerteté-

sére terjedelmi okokból nem vállalkozhatunk, igy az irodalomra utalunk. A dolgozat függeléke a szabályok szükségességének és elégségességének bizonyitását tartalmazza. A protokoll szintézis ezen megközelitésének alapvető korlátja az, hogy a szabályokat csak két entitásból (processből) álló rendszerre dolgozták ki, olyanra amelyben az entitások közötti kommunikációs csatornák FIFO (first in - first out) jellegüek.

A [Zafiropuló és m. 80] dolgozatban közölt szabályok n processre és nem szükségképpen FIFO csatornákra történő általánositását Sidhu adta meg a [Sidhu 82] dolgozatában.

2.1.3 Párbeszéd elmélet (Theory of colloquies)

A párbeszéd elméletet Le Moli publikálta korai cikkében [Le Moli 73]. Az elmélet két partner (un. interlocutor) közötti interakciók leirására alkalmas. Az interlocutorok kötött strukturáju absztrakt véges állapotu gépek. A 2.3 ábra bemutatja a két interlocutor input és output jeleit, összekapcsolásukat.



2.3 ábra Az összekapcsolt interlocutorok

Az interlocutor bemenő jeleit a

- másik partner üzenetei,
- az interlocutor felhasználójának parancsa,

τ a másik partner felé továbbitandó szöveg

jelek alkotják, mig az output jelek:

m üzenetek a partner felé,

t a partnertől kapott szöveg,

c az interlocutor felhasználója felé küldött parancsok.

Az interlocutor belső müködését, strukturáját a [Harangozó 78b],[Margitics 81] disszertációk részletesen elemzik magyar nyelven, igy bemutatásuktól eltekintünk.

Az [Alfonzetti és m. 79] dolgozat a párbeszéd elméletet használja az X.25 ajánlás DTE csomag szint formális leirására. A szerzők beszámolnak arról, hogy a protokoll formális megfogalmazása (minden verifikációs eljárás nélkül is) néhány kétértelmüség felfedezéséhez vezetett, mert rákényszeritette őket a teljes rendszer legappróbb részleteinek áttekintésére.

Le Moli gondolatát a [Danthine és m. 78] dolgozatban Danthine és Bremer fejlesztette tovább. Módszerüket end-to-end transzport protokollok leirására használták. A közleményben először az end-to-end transzport protokollok általános vonásait és az általuk nyujtott kommunikációs szolgáltatások leirását találjuk. A szerzők a transzport rétegtől két alap szolgáltatást várnak el, a réteg felhasználói processei közötti virtuális linkek létrehozását, megszüntetését és a létrehozott kapcsolaton keresztüli biztonságos adattranszfert. Modellükben egy virtuális link két végén egy-egy link-gép (link-machine, az interlocutor továbbfejlesztett változata) helyezkedik el. Feltételezik, hogy a réteg képes dinamikusan létrehozni és üzemeltetni az ilyen link gépeket abban az esetben, ha több virtuális link kialakitására van szükség. A link gép jelei a 2.4 ábrán láthatók.

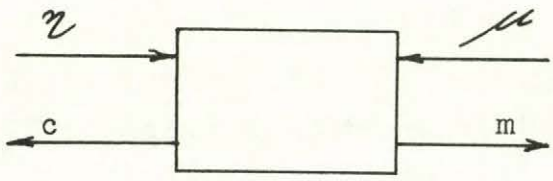
Az input jelek:

$\mu \in I_d$ (message envelopes)

$\zeta \in I_e$ (request blocks)

Az output jelek:

$m \in O_d$ (message envelopes)
$c \in O_e$ (response blocks)

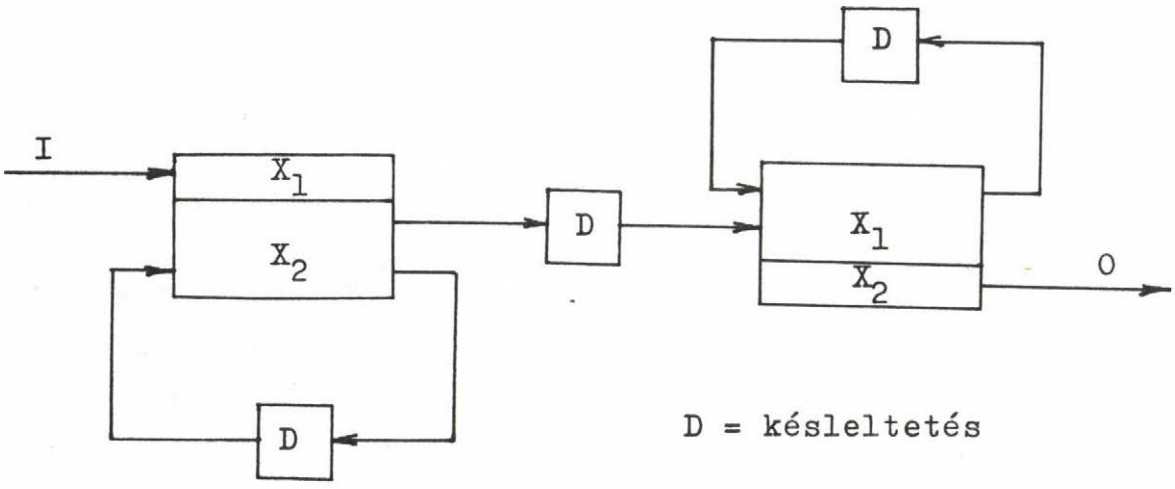


2.4 ábra A link gép jelei

Mint ahogy az a 2.4 ábráról leolvasható, a link gép Le Moli interlocutorának egyszerüsitett változata. A szerzők a link gép matematikai leirására a 2.1.1 fejezetben már ismertetett véges automatát használják. A jelölésbeli eltérések miatt a következőkben a [Danthine és m. 78] cikkben közölt jelöléseket alkalmazzuk. Az

(X, I, O, N, M)

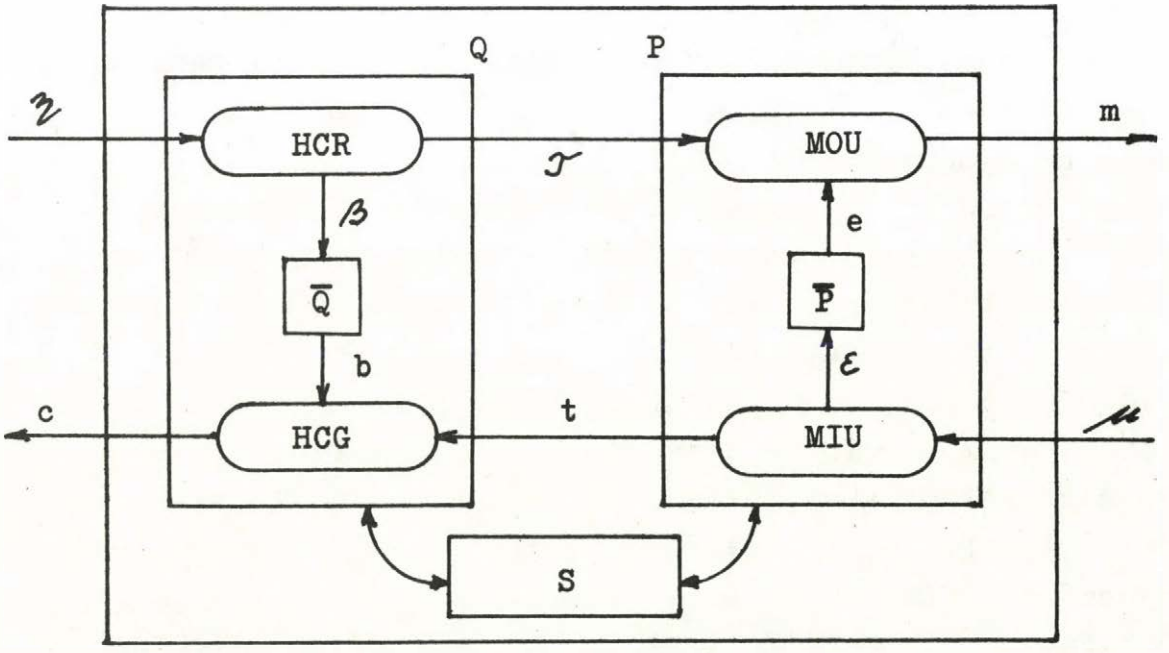
automata I (input) és O (output) jeleinek halmazát az $I=I_e \cup I_d$ és $O=O_e \cup O_d$ adják. Az N az átmenet, mig az M a kimenet függvény. A link gépet leiró véges automata dekompoziciójával két gépet, az un. alap automatát (basic automaton) és az un. kontext automatát definiálták.



2.5 ábra A kontext és alap automaták együttmüködése (vázlat)

X_1 az alap automata, X_2 a kontext automata állapotainak halmaza. A 2.5 ábra a rendszer müködésének vázlatát tartalmazza. Az input hatására megváltozik a kontext automata állapota (X_2) a basic automata X_1 állapotától függően, majd második lépésként az alap automata állapota változik és megtörténik az output jel létrehozása, amely függ a kontext automata állapotától is. A dolgozat második része protokoll verifikációs kérdésekkel foglalkozik. Definiálja a deadlock fogalmát és az elérhetőségi analizissel ekvivalens módszert közöl felfedezésükre. Megemliti, hogy a kontext automata helyesség ellenőrzése külön módszert igényel.

A párbeszéd elmélet tovább fejlesztésére Le Moli is kisérletet tett. Az általa kidolgozott un. "második párbeszéd elmélet" (Second Theory of Colloquies) az [Alfonzetti és m. 80] dolgozatban az X.25 3. szint megfogalmazását szolgálta. A második párbeszéd elmélet interlocutorának belső felépitését a 2.6 ábrán tüntettük fel. Ezen interlocutor a korábbihoz képest kevesebb input és output jellel rendelkezik.



2.6 ábra Az interlocutor belső strukturája

Az input jelek:

$\mathcal{U}$ üzenetek a másik interlocutortól,

$\mathcal{Z}$ parancsok az interlocutor felhasználójától.

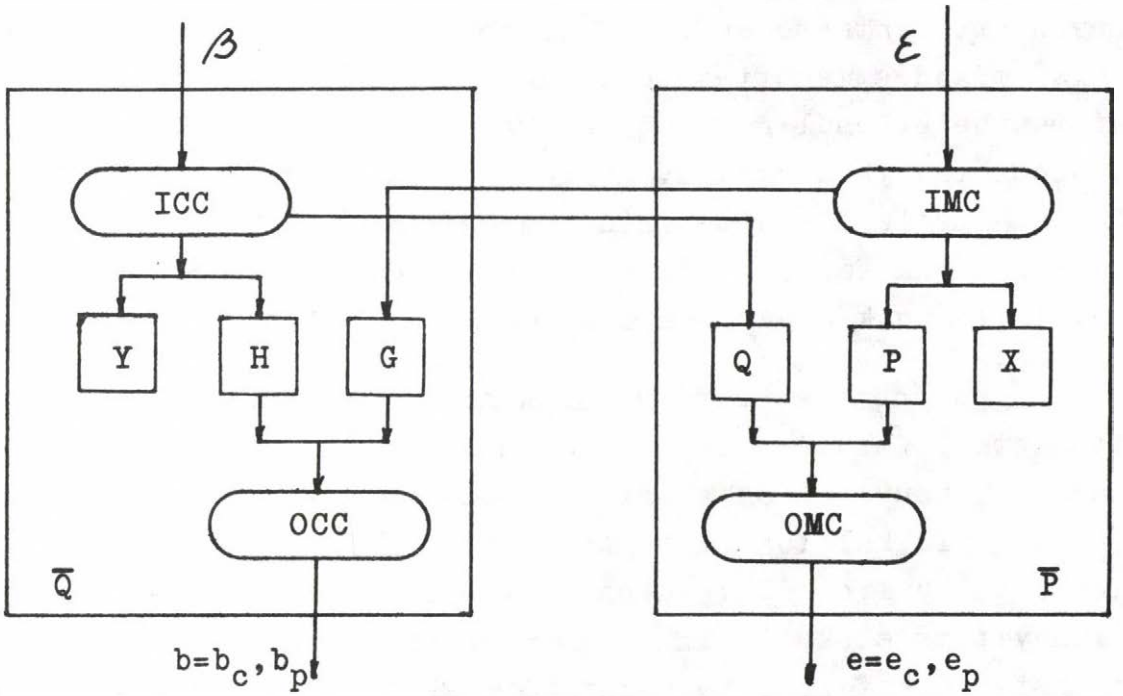
Az output jelek:

m üzenetek a másik interlocutor felé,

c parancsok az interlocutor felhasználójának.

Az üzenetek és parancsok két részből állnak, a szövegrészből (text) és a boritékból (envelope). A szövegrészt az interlocutor változatlanul továbbitja. A boriték operációs kódra és egyéb paraméterre tagolódik. Az interlocutor a boritékokban kapott információk alapján állapotátmeneteket hajt végre és előállitja az output jeleket. A Q magasabb szint felé mutatott interfész a HCR (Higher level Command Receiver) parancs fogadó részből, a $\overline{Q}$ egységből és a HCG (Higher level Command Generator) parancs generátorból áll. A HCR az $\mathcal{Z}$ parancsot szétválasztja a $\mathcal{T}$ szöveg és $\mathcal{B}$ boriték részre. A $\overline{Q}$ a $\mathcal{B}$ alapján előállitja a c kimeneti jel b boriték részét, majd a HCG a b boriték és t szöveg felhasználásával a c kimenő parancsot generálja. A P protokoll egység a két interlocutor közötti üzenetcserét szabályozza. A MIU (Message Input Unit) üzenetet fogadó egység $\mathcal{E}$ boriték és t szöveg részre tagolja a bejövő üzenetet. A MOU (Message Output Unit) a $\overline{P}$ által generált e boritékot és $\mathcal{T}$ szöveget m üzenetté formálva elküldi a másik interlocutor felé. Az S belső állaptváltozókhoz mind a Q, mind a P egység hozzáférhet. A $\overline{P}$, $\overline{Q}$ egységek belső felépitését tüntettük fel a 2.7 ábrán. A $\overline{P}$ és $\overline{Q}$ egységek külön kezelik a boritékok operációs kódját és az egyéb paramétereket. Az ICC (Input Command Context) és az IMC (Input Message Context) összehasonlitja a bejövő boritékok paramétereit az állapotváltozók egy részével az un. kontext változókkal, és esetleg megváltoztatják azok értékét. Az X és Y egységek az állapotváltozók másik részét képező un. fő bináris állapotváltozók értékét módositják. Az OCC (Output Command Context) és az OMC (Output Message Context) a b_c és e_c operációs kódokat kiegészitve a megfele-

lő b_p és e_p paraméterekkel, a b és e kimeneti boritékokat generálják. Az egységek közötti szinkronizációt szigoru- (itt nem tárgyalt) szabályok biztositják.



2.7 ábra A $\overline{P}$ és $\overline{Q}$ egységek

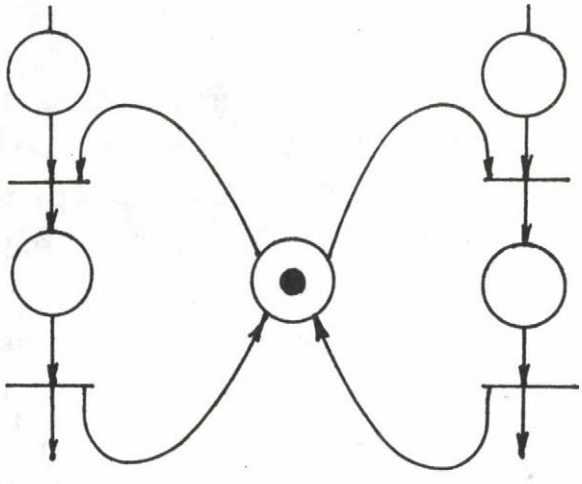
A belső egységek leirására különféle matematikai modelleket lehet felhasználni. Az [Alfonzetti és m. 80] dolgozatban például az ICC, IMC, OCC, OMC-t ALGOL szerü magasszintü programozási nyelvvel, mig a P, Q, G, H, X, Y egységeket pedig logikai mátrixokkal definiálták. A szerzők a dolgozatban kifejtik, hogy a második párbeszéd elmélet segitségével egyszerübb, áttekinthetőbb leirását kapták az X.25 3. szintjének, mint az első elméletet felhasználó [Alfonzetti és m. 79] dolgozatban. Mindez az uj interlocutor erősen moduláris felépitésének köszönhető.

2.1.4 Gráf modellek

A kommunikációs protokollok leirására gyakran használ-

nak gráf modelleket. A gráf formalizmusok legnagyobb előnye (és egyben alapvető korlátja) az, hogy a rendszer működéséről grafikus képet nyujtanak. Kis vagy közepes bonyolultságu probléma (pl. protokoll) gráfja kis méretü, könnyen áttekinthető és igy jelentős segitséget tud nyujtani a belső müködés megértésénél. Nagy méreteknél azonban a gráf menthetetlenül elveszti ezeket a jó tulajdonságait. Ez esetben külön technikákra, strukturálási módszerekre van szükség a megfelelő áttekinthetőség fenntartásához. A gráf modellek másik fontos előnye az, hogy "természetes" formalizmust kinálnak a párhuzamosságok leirására.

Legkorábban a Petri és UCLA gráfokat használták ezen a területen. Mivel a két modell szigoru ekvivalenciáját később bebizonyitották (Postel), ezért a következőkben csak a Petri gráfokkal fogunk foglalkozni. A Petri gráf gráfpontokból, un. helyekből (places), átmenetekből (transitions) és irányitott élekből áll. A 2.8 ábrán egy kis gráfrészletet tüntettünk fel, a helyeket körökkel, az átmeneteket vonalkával jelölve.



2.8 ábra Petri háló részlet (erőforrás)

A gráf helyeit feltételeknek, az átmeneteket eseményeknek megfeleltetve a párhuzamos rendszerek természetes modelljét kapjuk. A Petri hálóban bárcák (tokens) mozognak. Ha egy gráfpontban megjelenik egy bárca, akkor az, az illető hely-

hez rendelt feltétel teljesülését jelenti. A bárcák Petri hálóban történő mozgása szigoru szabályokhoz van kötve. Egy bárca egy átmeneten keresztül csak akkor haladhat át, ha az átmenet minden input helyén (azaz azokon a helyeken, amelyből irányitott él vezet az átmenethez) van bárca. Az áthaladást az átmenet tüzelésének (firing) nevezik. Tüzeléskor az átmenet input helyeiről egy-egy bárca átkerül az output helyekre, eggyel megnövelve az ott lévő bárcák számát. (Megjegyezzük, hogy Petri eredeti konstrukciójában egy helyen egy időben csak egy bárca helyezkedhetett el.) A modell fenti általánositását (több bárca egy helyen) csak később végezték el.

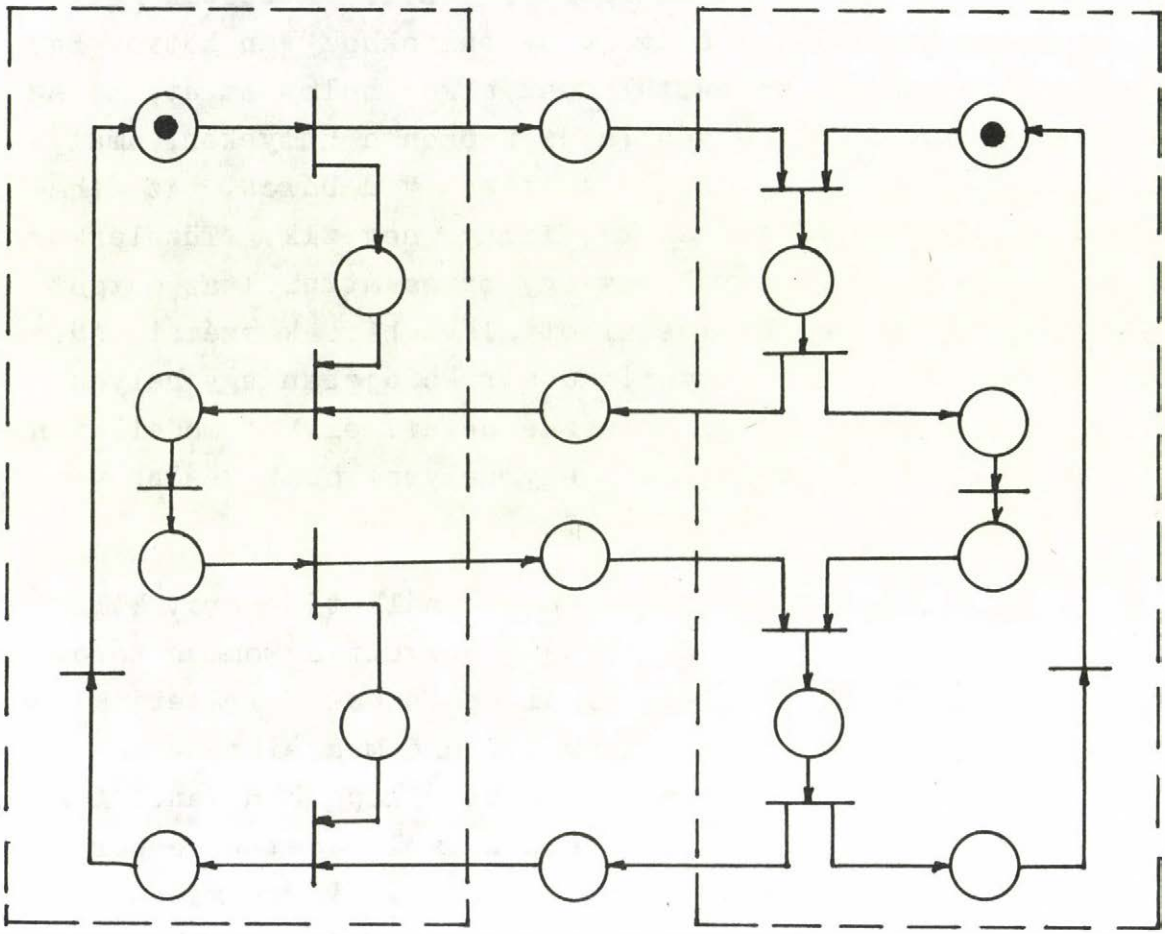
A Petri hálót sokféleképpen használhatjuk protokollok leirására. Például ha a Petri háló helyeit a kommunikáló protokoll entitások (állomások) állapotainak feleltetjük meg, akkor a bárcák ilyen helyen történő megjelenése azt jelzi, hogy az adott entitás az adott állapotban van. A helyek csomag tipusokat is jelölhetnek, mely esetben a bárcák az adott tipusu csomagok számát jelentik. Ilyen módon a helyek a kommunikációs mediumok modellezésére is alkalmasak. A 2.9 ábra egy egyszerü protokoll Petri hálóját tartalmazza. A háló jobb és bal oldalán a protokoll entitásoknak megfelelő, mig középen a mediumot reprezentáló helyek találhatók. A bárcák kezdeti eloszlását szintén feltüntettük.

A Petri háló nem grafikus reprezentánsát az

$$I_1, I_2, \ldots I_n \longrightarrow O_1, O_2, \ldots O_m$$

formáju átmenet szabályok táblázatos felsorolásával kaphatjuk meg. Az I_i ($i = 1..n$) az input helyeket, az O_j ($j = 1..m$) pedig az output helyeket jelöli. Az átmeneti szabály egy átmenethez tartozó input és output helyeket rendeli egymáshoz.

A Petri hálót használó protokoll leirási mód lehetőséget nyujt bizonyos verifikációs módszerek alkalmazására. A verifikációs lehetőség a Token Machine fogalmára épül. A



2.9 ábra Egyszerü protokoll Petri hálója

A Petri háló állapotát a bárcák eloszlása határozza meg. A Token Machine egy olyan gráf, amelynek pontjai a Petri háló állapotait reprezentálják. Irányitott élei segitségével végigkövethető a rendszer globális müködése. A Token Machine tehát az elérhetőségi gráf fogalmához hasonló; alapvetően a deadlock szituációk felderitését könnyiti meg.

A Petri háló korábban már emlitett hátránya a korlátozott leiróképességben és a gráfpontok gyors "szaporodásában" rejlik (hasonlóan az egyszerü véges automatás leirási módhoz). Merlin a Time Petri Net fogalmának bevezetésével próbálta gazdagitani az egyszerü Petri háló leiró képességét [Merlin 76a],[Merlin és m. 76b]. A Petri háló átmenetének

tüzelése a korábban emlitett tüzelési feltétel bekövetkezte után bármikor megtörténhet. A Time Petri Net-ben az átmenethez két idő adat van hozzárendelve. Az átmenet a tüzelési feltétele teljesülése után legkorábban t^{min} idő mulva tüzelhet, de a tüzelés legfeljebb t^{max} idő alatt biztosan végbemegy.

A Petri gráfoknak számos egyéb továbbfejlesztett változata létezik. (E-gráf, makro-gráf, numerikus Petri háló stb.) Ezen gráfok kommunikációs protokollok leirására történő felhasználását a [Margitics 81] tanulmány részletesen elemzi, ezért bemutatásuktól eltekintünk, helyette a fenti munkára utalunk

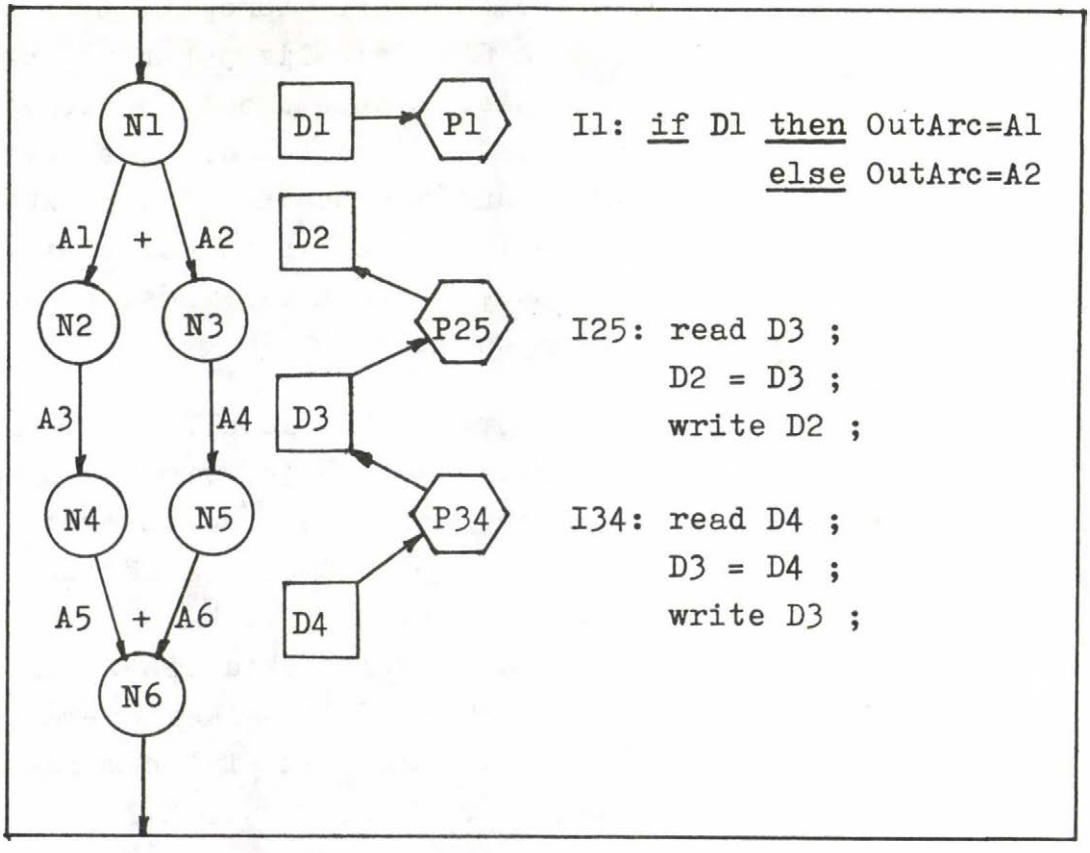
Az UCLA gráfok továbbfejlesztésének tekinthető az UCLA-n kidolgozott GMB (Graph Modell of Behavior) [Razouk és m. 80], [van-Mierop 79]. A SARA (System Architects' Apprentice) általános célu számitógéppel támogatott tervező rendszer segitséget nyujt a GMB formalizmusban megfogalmazott rendszerek (pl. kommunikációs protokollok) elemzéséhez. A SARA egy ilyen felhasználásáról számol be a [Razouk és m. 80] közlemény az X.21 ajánlás kapcsán. A következőkben röviden bemutatjuk a GMB-t. A GMB a protokollokat (rendszereket) három szempontból modellezi: a vezérlés, az adatáramlás és interpretáció szempontjából.

A "vezérlés folyam" (control flow) modellezésére a "vezérlés gráfot" (Control Graph) használja, amely lényegében egy UCLA gráf. A gráf csomópontjai az eseményeket (számitási lépéseket), mig az irányitott élek a köztük lévő precedencia relációkat modellezik. A gráfban a Petri hálóhoz hasonlóan bárcák jelzik a vezérlés aktuális állapotát. A gráf csomópontjaiba befutó élek között logikai operációk (AND (*) ; OR (+)) értelmezhetők, amelyek (itt nem részletezhető módon) befolyásolják a bárcák mozgását.

A GMB második része a rendszer adat folyamát (data flow) modellezi az un. adat gráffal (Data Graph), az input-

tól a feldolgozáson keresztül az outputig. Az adat gráf adat halmazokból (négyszög), vezérelt (hatszög) és nem vezérelt (háromszög) processzorokból (számitási pontokból) és irányitott élekből áll. A processzorok az adat halmazokon végeznek számitási müveleteket. A vezérelt (controlled) processzorok müködésüket csak akkor kezdik meg, ha a hozzájuk rendelt vezérlés gráf pontban megjelenik egy bárca. A müködés befejezésével a vezérlés gráfban a szóban forgó bárca tovább haladhat. A nem vezérelt processzorok aktivizálása független a vezérlés gráftól,(az inputjaik megváltozása aktivizálja őket). Az irányitott élek a processzorok és adat halmazok közötti adatáramlás irányát mutatják.

A GMB interpretációs részében az adathalmazok szintaxisának és a processzorok végezte eljárások megadása történik valamilyen magasszintü nyelven (pl. PL/1).



2.10 ábra Egyszerü rendszer GMB megfogalmazása

A GMB illusztrálásaként a 2.10 ábrán egy egyszerü rendszer vezérlés és adat gráfját mutatjuk be. Az interpretációs részben input és output utasitásokat, értékadást és a vezérlésre vonatkozó speciális utasitásokat tartalmazó programrészt tüntettünk fel.

A GMB modell vizsgálatára a SARA egy automatikus vezérlés folyam analizátort (control flow analyzer) bocsájt a felhasználók rendelkezésére [Razouk és m. 80]. A vezérlés folyam analizátor állapot összevonásokat végez és lehetőséget teremt az elérhetőségi analizis végrehajtására igen fejlett szoftver támogatás keretében.

2.1.5 Bochmann hibrid modellje

Bochmann és Gecsei a [Bochmann és m. 77a] dolgozatban kidolgozták a protokoll specifikáció egyesitett módszerét (unified method). Akkoriban már erősen érezhetővé vált, hogy a protokoll specifikációs módszerek két fő irányba fejlődnek. Az egyik irányt az állapotátmenetet felhasználó, mig a másik irányt a programozási nyelven alapuló módszerek (2.2 fejezet) képviselik. A szerzők felismerték, hogy a két féle megközelités a protokollok más-más tulajdonságainak formalizálását könnyiti meg. Programozási nyelv segitségével könnyen kifejezhetők azok a protokoll vonások, amelyek nagy számu állapotátmenet felvételét igényelnék az állapotátmenet leirás keretében (pl. sorszámozás), mig a vezérlés áttekinthető megfogalmazását a programozási nyelvekkel szemben inkább az állapotátmenet módszer támogatja. A két formalizmus tehát előnyösen kiegészithetné egymást. Verifikációs szempontból azonban a két megközelités jelentősen eltér egymástól. A korábban már tárgyalt állapotátmenetet felhasználó specifikációs módszerek esetén az elérhetőségi vizsgálat az alapvető verifikációs lehetőség, mig a programozási nyelvek program helyesség bizonyitó eljárásokat igényelnek.

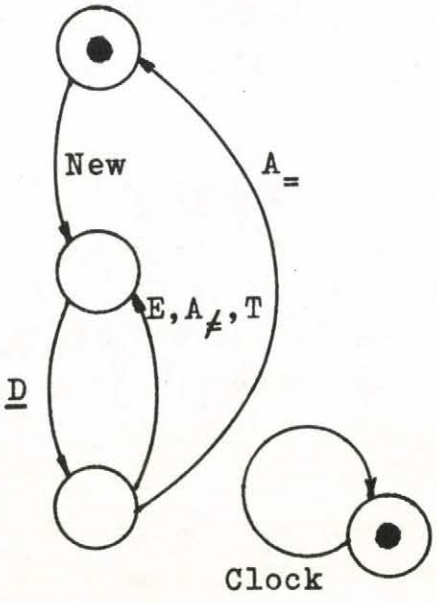
A dolgozatban leirt hibrid (egyesitett) módszer a [Keller 76] javaslatra épül. Keller a Petri háló (2.1.4 fejezet) változókkal történő kiegészitését javasolta a paralell programok formális leirására. A Petri háló minden átmenetéhez egy a változóktól függő engedélyező predikátumot (enabling predicate) és egy akciót (action) rendelt. Az akció a változón végzett müveleteket irta le. A rendszer állapotát a Petri háló különböző helyein lévő bárcák (token) és a változók értékei alkották. A modell müködése a következő volt: a rendszer egy átmenete akkor vált végrehajthatóvá (enabled), ha teljesült a Petri háló már megismert tüzelési feltétele (minden input gráfpontban van bárca) és a hozzá tartozó engedélyező predikátum értéke igaz volt. Az átmenet során a bárcák a Petri háló szabályainak megfelelően mozogtak és az átmenethez rendelt akció végrehajtódott.

Bochmann és Gecsei a Keller modellt az elosztott rendszerek leirására tették alkalmassá. A Keller modellben bármely akció bármely változó értékét megváltoztathatta és bármelyik változó szerepelhetett akármelyik engedélyező predikátumban. Az elosztott rendszerek egymástól esetleg nagy távolságban elhelyezkedő kommunikáló alrendszerekből állnak. Az ilyen rendszerek leirásánál egészen természetes módon vetődik fel az az igény, hogy egy alrendszer (lokális) változóját csak az adott alrendszer (lokális) átmenete változtathassa meg. A szerzők tehát kiegészitették a Keller modellt az egymással kommunikáló alrendszerek fogalmával. Felfogásuk szerint az S rendszer S_1, S_2, ... S_n alrendszerekből áll. Az S_i alrendszer lokális változóinak halmazát az X_i alkotja. Az S_i alrendszer (lokális) predikátumai és akciói ezen X_i halmazon értelmezettek. Az alrendszerek közötti kommunikáció leirására bevezették a paraméterezett "távolról kezdeményezett müveletet" (distantly initiated action). Ezen müvelet a közönséges akciókhoz hasonlóan a lokális változók értékét módositja, de végrehajtása nem átmenethez kötött. A végrehajtás egy távoli alrendszer kezdeményezése (initiate

utasitás kiadása) után véges időn belül megtörténik. Egy adott alrendszer "távolról kezdeményezett müveletei" kölcsönös kizárásban futnak. Az alrendszerek közötti kommunikáció formája az üzenet csere, amely a "távolról kezdeményezett müvelet" paraméterein keresztül materializálódik. Az üzenet küldés a "távolról kezdeményezett müvelet" inicializálásának, mig az üzenet fogadás a müvelet végrehajtásának felel meg. Az inicializáló utasitás

initiate (name, p_1, p_2, ... p_n)

alaku, ahol a name a "távolról kezdeményezett müvelet" neve, a p_i (i = 1..n) a paraméterek. A szerzők a változó deklarációkat, a predikátumokat, az akciókat PASCAL-hoz [Wirth és m. 74] hasonló nyelven fogalmazták meg, a Petri hálót a bárcákkal grafikusan ábrázolták (2.11 ábra).



átmenet	eng. predikátum	müvelet
New	true	new(data); seq:=seq+1(mod2);
D	true	initiate(transD, seq, data);
$A_=$	ack=seq	
$A_{\neq}$	ack=seq+1 (mod2)	
E	ack=error	
T	tout=true	
Clock	true	time:=time-1; if time=0 then tout:=true;

Változók
seq: (0,1)
ack:(0,1,error,none)
data:
tout: boolean
time: integer

távolról kezdeményezett müvelet
transA(p:(0,1))
case transmission of
correct: ack:=p;
erroneous: ack:=error;
loss:;

2.11 ábra A Bitalternáló protokoll hibrid modellje

Ezen formális modell keretében egy adott pillanatban a rendszer állapotát a bárcák eloszlása, a változók értékei és az inicializált, de még nem végrehajtott "távolról kezdeményezett müveletek" képezik. A rendszer állapot definiálására verifikációs célból került sor. A dolgozatban kidolgozott protokoll verifikációs módszer az állapotelérhetőségi analizisre épül. Az alrendszerek definiciója alapján felépitik a müködésre jellemző elérhetőségi gráfot, amely segitségével kimutatják a deadlock helyzeteket és ciklusokat. A specifikáció parciális helyességének bizonyitását globális predikátumok igazolásával végzik. Teljes állapotnak (complete state) nevezik az olyan állapotot, amelyből a globális predikátumok érvényessége következik. A protokoll teljes helyessége alatt a létező teljes állapot liveness tulajdonságát (8. fejezet) értik, azt, hogy a teljes állapot véges időn belül "sorra kerül" a protokoll müködése során. A hibrid specifikációs és verifikációs módszer illusztrálására a Bitalternáló protokoll leirását és helyesség ellenőrzését végezték el.

A [Bochmann és m. 77b] közlemény a hibrid specifikációs módszer HDLC protokollra történő alkalmazását tartalmazza, kiegészitve a formalizmust néhány uj gondolattal. A protokoll rendszert alkotó állomások (a korábbi cikkben alrendszerek) komponensekből épülnek fel. A komponensek program változókból és állapotátmenet diagramból állnak[1]. Az állomások közötti kapcsolat (kommunikáció) megvalósitását a korábban már ismertetett "távolról kezdeményezett müvelet" teszi lehetővé. A dolgozatban egy állomáson belüli komponensek között három féle kapcsolat tipust definiáltak.

[1] A protokoll modell ilyen megváltoztatása valószinüleg annak a következménye, hogy a bonyolultabb protokoll csak erősen moduláris megfogalmazásban áttekinthető; korábban az állomás (alrendszer) csak egy komponensből állt.

A párhuzamosan független komponensek (paralell independent component) állapotátmeneteiket egymástól függetlenül (paralell módon) végzik, melynek során a müveletek segitségével értéket adnak a saját és esetleg más komponensekkel közös un. osztott változóknak. A közös (osztott) változókhoz való hozzáférés a komponensenkénti kölcsönös kizárásban megy végbe, vagyis egy időben csak egy komponens változtathatja meg az osztott változó értéket.

Az X komponens hierarchikusan függ az Y komponenstől (hierarchically dependent component), ha X csak akkor inicializálódik[2], ha Y valamely meghatározott állapotba kerül és ha Y kikerül ebből az állapotból akkor X "passziválódik"[3].

Két komponens direkt csatolt (direct coupling), ha a két komponens állapotátmenetei között van legalább egy direkt csatolt állapotátmenet pár. A direkt csatolt átmenetek csak párhuzamosan (szimultán) hajthatók végre, ami azt jelenti, hogy az átmenet pár végrehajtása az átmenetek előfeltételeinek együttes teljesülése esetén lehetséges.

Az összefoglaló jellegü [Bochmann 80] cikk első része a [Bochmann és m. 77b] dolgozat hibrid protokoll leirási módszerét mutatja be, példákon illusztrálva az elmondottakat. A második rész a protokoll szolgáltatás specifikáció kapcsán felmerülő különféle kérdésekkel foglalkozik.[4] Megkülönbözteti a kommunikációs szolgáltatás lokális és globális tulajdonságait, bár maga is elismeri a szétválasztás kissé mesterkélt voltát, ugyanis a globális tulajdonságok magukban foglalják a lokálisakat is. A szolgáltatás lokális tulajdonsá-

[2]A komponens inicializálása a komponens változók kezdeti értékeinek beállitását és a bárca kezdő helyzetbe hozását jelenti.

[3]A passziválódás a komponens müködésének befejeződését jelenti.

[4]Korábban már beszéltünk a szolgáltatás specifikáció és a protokoll specifikáció szétválásáról.

gai a szolgáltatást nyujtó entitás és a szolgáltatásokat igénybe vevő felhasználó (entitás) közötti interakciókat jellemzi, figyelmen kivül hagyva azt, hogy mi történik a kommunikáció két végpontja között. A globális jelző az un. end-to-end szolgáltatás tulajdonságokra vonatkozik.

A szolgáltatás specifikáció egyik legfontosabb része (lokális tulajdonság) a felhasználó és a rendszer közötti interfész leirása. Az interfész megadására a szerző bevezeti a "direkt csatolt interfész" (directly coupled interface) fogalmát, amely a felhasználói és a protokoll entitás közötti direkt csatolást jelenti. A direkt csatolt állapotátmenet párokat táblázatos felsorolással adja meg. Az interfész egyszerüsitésére absztrakciós lépéseket használ, amelynek alapja az információ rejtés elve. A felhasználói entitásnak ugyanis nem kell tudnia arról, hogy milyen a protokoll belső müködése, felépitése. A direkt csatolt interfész átmenetek összevonásával a szolgáltatás primitivek (service primitives) megalkothatók. A szolgáltatás primitivek kezdeményezőjének jelzésére a "↑" és "↓" jeleket vezeti be. A "↑" a szolgáltató entitás, a "↓" a felhasználó kezdeményezte primitivek jelzésére szolgál. A szolgáltatás primitivek végrehajtása során az interakcióban lévő entitások közötti pótlólagos kommunikációt a szolgáltatás primitivekhez csatolt paraméter halmazzal oldja meg. A paraméterátadáskori információ áramlás irányának jelzésére a "⟶" és "⟵" jelöléseket alkalmazza. A szolgáltatás primitivek felfüggeszthetősége (discontinuation of service primitives) a komponensek közötti hierarchikus kapcsolatok megnyilvánulása interfész szinten. A szerző foglalkozik az ilyen módon megadott interfész implementációjának kérdésével is. Az implementáció során olyan problémákat is figyelembe kell venni mint például az interfészt megvalósitó entitások közötti adatfolyam vezérlés (flow control) stb.. Megemliti, hogy a szolgáltatást nyujtó és a felhasználó entitás közötti interfész implementációja sok esetben üzenet sorok segitségével történik

(pl. [Böszörményi 81]).

A kommunikációs szolgáltatás globális tulajdonságát a szolgáltatás primitivek végponti végrehajtási sorrendje képezi. Elosztott rendszerekben az események sorrendjének megállapitása nem triviális feladat. A szerző globális, abszolut időt feltételezve a formális nyelvek produkcióihoz hasonló átirási szabály rendszert alkalmaz az események sorrendjének rögzitéséhez.

A következő részben Bochmann felsorolja a kommunikációs szolgáltatás specifikáció fő elemeit. A szolgáltatások természetes emberi nyelvi rögzitése áttekintést ad a kérdéses protokoll réteg céljáról, müködéséről. A szolgáltatás primitivek egyenkénti felsorolása a primitivek szintaktikai leirását (nevét, paraméterek számát, tipusát) tartalmazza. A szolgáltatások lokális tulajdonságait a primitivek végrehajtási sorrendjének meghatározása alkotja. Ha a primitivek végrehajtási sorrendjét (a megfelelő paraméter cserével) a kommunikációs végpontok szempontjából rögzitik, akkor az a globális tulajdonságok megadása. A szolgáltatások mennyiségi követelményeinek meghatározása (késleltetés, átbocsájtó képesség) zárja a szolgáltatás specifikáció főbb elemeinek felsorolását.

A közlemény utolsó részében a szerző néhány megjegyzést füz a korábban napvilágot látott verifikációs módszerek némelyikéhez [Zafiropuló és m. 80],[Bochmann 78].

2.2 Programozási nyelveken alapuló módszerek

A programozási nyelven alapuló protokoll definiálási módszerek alapja az, hogy a programozási nyelvek kényelmes eszközöket biztositanak a protokoll adatszerkezeteinek és azokon értelmezett müveletek leirásához. A protokollok algoritmuskénti megfogalmazása nem csak azzal az előnnyel jár, hogy a definició ilyen módon tömör és világos, hanem még nagymértékben megkönnyitheti a protokoll implementációját is. A definició implementáció közelisége az alkalmazott nyelv absztrakciós szintjétől függ. A módszerek hátrányaként szokták megemliteni azt az ellenvetést, hogy az ilyen specifikációkban keverednek a lényegtelen programozás technikai megoldások a lényeges algoritmusokkal.

A legtöbb programozási nyelvi protokoll specifikációban a protokoll entitásokat különálló program modulok, processek reprezentálják, a közöttük meglévő kommunikációt pedig osztott változókkal vagy üzenet cserével modellezik. A protokoll specifikáció során jól alkalmazhatók a szoftver tervezés különféle módszerei (hierarchikus, top-down, bottom-up tervezés) [Molnár 81].

A [Bochmann 75] dolgozatban a HDLC protokoll néhány tulajdonságát elemzi a szerző. A protokoll definiálására PASCAL szerü nyelvet használ. A HDLC protokoll rövid szöveges bemutatása után felrajzolja a protokollt alkotó entitások vázlatos képét. Az entitások leirására (állapotuk definiálására) változókat (számlálókat, üzenet pufferelléshez használt adatstrukturákat) alkalmaz. Az entitások algoritmikus müködését a változókon értelmezett PASCAL utasitások sorozatával (programmal) adja meg. A komponensek (állomások) közötti kommunikációt kötött strukturáju üzenetek (rekordok) cseréjével modellezi. A protokoll helyes müködését állitások, tételek bizonyitásával végzi. Bebizonyitja, hogy véges puffer kapacitás esetén is jól müködik a protokoll, amely az implementálhatóság alapvető feltétele. A

protokoll implementálásához a CONCURRENT PASCAL [Brinch-Hansen 75] nyelv process és monitor fogalmait használja és kimutatja az ilyen magasszintü, jól strukturált párhuzamos nyelv előnyeit. A dolgozat vitathatatlan érdeme az, hogy a szerző egységes szemléletmóddal közeliti meg a kérdést, nem különálló, hanem szorosan összefüggő folyamatnak tekinti a protokoll definiálásától kiinduló, a verifikáción keresztüli, implementációval végződő utat.

A [Stenning 76] közleményben adattranszfer protokoll PASCAL nyelvü specifikációját találjuk. A szerző élesen szétválasztja a virtuális kapcsolat (virtual connection) szolgáltatást nyujtó host-host szintü protokoll két funkcióját a kapcsolat létesités, bontás és az adatszállitási funkciókat. Dolgozatában csak az utóbbival foglalkozik. Az általa vizsgált protokoll az alacsonyabb hálózati szintek közremüködésével egy irányu adatszállitási szolgáltatást nyujt, kiküszöbölve az alhálózat feltételezett hibáit (csomag vesztő képesség, duplikálódás, sorrendcsere stb.). A protokoll pozitiv nyugtázási mechanizmust és timeout esetén ujraküldést alkalmaz. A megfogalmazásban a szerző két entitást, az üzenet küldő és fogadó entitást használ. Mindkettő egy-egy PASCAL nyelvü process, amelyek osztott adatokon keresztül kommunikálnak. A strukturált, moduláris megfogalmazás érdekében a processek procedurákból álló un. prefixszel rendelkeznek. (Egy adott process csak a saját prefixének eljárásait használhatja.) Mindezek eredményeképpen a specifikáció jól olvasható, könnyen áttekinthető. A protokoll helyes müködéséről a Hoare féle programhelyesség bizonyitó eljárás alkalmazásával győződik meg. Csak parciális verifikációt végez.

A [Stenning 79] publikációban a fenti protokoll mellett még az un. "three-way handshake" kapcsolat létesitési és bontási protokoll (connection control protocol) specifikációja és vizsgálata is megtalálható.

Az [Ethernet 80] az Ethernet lokális hálózat fizikai és adatkapcsolati rétegének specifikációját tartalmazza. A dokumentum az adatkapcsolati réteg formális leirásához az un. Procedurális Modellt (Procedural Model) használja. A Procedurális Modell processekből és az általuk használt eljárások PASCAL nyelvü megfogalmazásából áll. A PASCAL nyelv változatos adattipus definiálási lehetősége kiválóan alkalmazható a különféle csomag, keret formátumok definiálására. A cikk az eljárások hivási lánca végigkövetésének megkönnyitésére irányitott gráfot (eljárás hivási gráf) közöl. A processek közötti szinkronizációt közös változókkal oldja meg. A specifikáció behatóbb vizsgálatából kitünik a réteg hardver megvalósitásának szükségessége.

A [Brand és m. 78] publikációban a szerzők PASCAL szerü nyelv segitségével irták le a protokollokat és a verifikációhoz a szimbolikus végrehajtás módszerét használták. Módszerük kidolgozását mikroprogramozott számitógép központi egysége és az I/O interfész közötti kommunikáció problémái motiválták. Dolgozatuk első részében egy triviális példa segitségével a szimbolikus végrehajtás módszerét illusztrálták. A következőkben a PASCAL szerü nyelv két utasitással (delay, wait ϕ) történő kiegészitését tárgyalták. Végül az adott leirási mód esetére a szimbolikus végrehajtás menetét (algoritmusát) irták le. Módszerük a deadlock helyzetek, tempo-blocking-ok felderitését teszi lehetővé. A publikált eljárás részletes magyar nyelvü ismertetése a [Harangozó 78b],[Margitics 81] disszertációkban hozzáférhető.

Figyelmet érdemel a [Bochmann és m. 79] dolgozat, amelyben az X.25 specifikációjáról és implementációjáról számolnak be a szerzők. A protokoll rendszer definiálását a 2.1.5 fejezetben már bemutatott hibrid módszer segitségével végezték el. A CONCURRENT PASCAL nyelvü implementáció a specifikációból közvetlenül származtatható, ugyanis a szerzők kidolgozták azokat a transzformációs szabályokat, amelyek leképezik a specifikáció komponenseit a CONCURRENT

PASCAL nyelv process, class és monitor elemeire. A részletes magyar nyelvü ismertetés a [Böszörményi 79] OMFB tanulmányban megtalálható.

A [Böszörményi 81] tanulmányban külön fejezet foglalkozik az Akadémiai Számitógép Hálózat virtuális terminál protokollja MODULA-2 nyelvü modelljének megvalósitásával. A modell célja a protokoll szöveges definiciójának formális megfogalmazása, egyértelmüvé tétele. A modell elméletileg egy implementáció, azonban a környezet (alsó és felső protokoll szintek), interfészek szimulációjával a szerző egy kisérleti szemléltető és tesztelési eszközt hozott létre, amely lehetővé teszi a hálózat belsejében áthaladó információk megjelenitését, különféle hiba szituációk létrehozását és annak a megfigyelését, hogy a rendszer hogyan viselkedik e hibák megjelenésekor.

Mind az a korábbiakból is kitünhet a magasszintü és különösen a nagyon magasszintü nyelvek (VHLL) protokoll leirásra történő alkalmazása egyre inkább elmossa a specifikáció és implementáció közötti különbséget. Ebbe az irányba tett lépésnek tekinthetjük a kifejezetten protokoll specifikációs célra kifejlesztett magasszintü nyelvek megjelenését is. Mivel e munka későbbi részeiben részletesen elemezzük a protokoll specifikációs nyelvek fő tulajdonságait, ezért a fejezet hátra lévő részében csak felvillantjuk az ide vonatkozó közleményeket.

A [Piatkowski 79] az ISO-OSI Referencia Modelljének formális definicióját tartalmazza. A leirási mód az un. állapot orientált specifikáció. A Referencia Modellt különböző részletességü blokkdiagramok segitségével ábrázolták, majd megadták az egyes blokkok programozási nyelvi kifejtését. Az alkalmazott programozási nyelv a PASCAL szintaktikáját követi, de uj szemantikai elemeket is magában foglal. Lehetőség van a blokkok input/output viselkedésének egyértelmü leirására, véges állapotu gépek nyelvi defi-

niálására. A megközelités hibrid jellegü, amely megpróbálja a véges állapotu gépet alkalmazó leirási módot összeegyeztetni a procedurális specifikációs móddal. A nyelv felhasználója szabadon megválaszthatja azt, hogy az adott blokk leirásához melyik formát választja. A leirás lehetőséget teremt a specifikált rendszer müködésének szimulálására, géppel támogatott analizisére.

Ugyancsak a Piatkowski kidolgozta uton indultak el a [Schultz és m. 80] közlemény szerzői, amely az IBM SNA architekturájának formális leirásához és validációjához vezetett. (A korábban IBM dolgozó Piatkowski dolgozta ki az állapot orientált leirási módot, amely a FAPL protokoll specifikációs nyelv alapját képezte.) A közlemény első részében az SNA szolgáltatásait, réteges strukturáját mutatják be a szerzők. A következő részben az állapot orientált specifikáció lényeges kérdéseit elemzik. A leirás kiinduló pontja a rendszer blokk diagramja(i), amelye(ke)t fokozatosan finomitva végül olyan blokk diagramhoz lehet jutni, amely blokkjainak leirását már véges automatákkal meg lehet oldani. A blokk diagramok, állapotátmenet gráfok mellett a leirás számlálókat, regisztereket, sorokat és egyéb táblázatokat is használ. Igy lényegében létrejött az SNA egy "emberi fogyasztásra alkalmas" (human-executable) modellje, un. meta implementációja. A következő lépés a géppel feldolgozható (machine-executable) specifikáció megalkotása volt. Ez vezetett el a FAPL specifikációs nyelv kialakitásához.

A FAPL (Format and Protocol Language) a PL/1 programozási nyelv bővitett, kiterjesztett változata. A kiterjesztést három fő irányban végezték el. Az első irányt az adat entitások (egységek) létrehozása és megszüntetése, a lista és sor kezelés megkönnyitése jelentette. A második fontos bővités a véges állapotu gépek definiálása lehetőségének megteremtése volt. A véges állapotu gépek definiálása (nevének, állapotátmenet mátrixának megadása) és müködtetése (állapotátmenet kényszerités) mellett a nyelv lehetőséget nyujt a

gép állapotának, attributumainak lekérdezésére, validációs, tesztelési célu állitások megfogalmazására. Harmadsorban egységes hivatkozási rendszert vezettek be a különböző protokoll szintek azonos kialakitásu protokoll modulljaira. A FAPL nyelvü specifikációt preprocesszor forditja le a PL/1 elemeire, megteremtve ezzel a modell végrehajtatásának lehetőségét. A közlemény utolsó részébe a szerzők az SNA "data flow control" rétegének validációjáról szóló beszámolót helyezték el. Az alkalmazott verifikációs eljárás az állapot perturbációs módszer (2.1.2 fejezet) volt.

Az [ISO 81b] munkaanyag egy a PASCAL nyelvre épülő specifikációs nyelvi javaslat vázlatát foglalja magában. Az FDT (Formal Description Techniques) javaslat alapja az un. bővitett állapotátmenet modell (extended state transition model), amelyet megpróbáltak beleilleszteni a PASCAL nyelv konvencionális elemei közé. A munkaanyag csak vázlatosan tárgyalja az uj nyelvi konstrukciókat, rövid szöveges leirás után szintaktikai javaslatokat tesz. Az anyag a továbbgondolkodásra sarkalló nyitott problémák felsorolásával végződik. Az FDT kisérleti felhasználásáról tudósit a [Leveille és m. 82] working paper.

A [Schwabe 81] munkában a szerző a SPEX protokoll specifikációs nyelvet ismerteti. A protokoll SPEX nyelvü leirását algebrai adattipus specifikációs formalizmusra forditja le. A protokoll absztrakt adattipuskénti megfogalmazása azzal az előnnyel jár, hogy alkalmazhatóvá válik az UCLA-n kidolgozott AFFIRM absztrakt adattipusok félautomatikus verifikációját végző rendszer protokoll verifikációs célokra. A módszerrel egy kapcsolat felépitési és egy elosztott adatbázisban használt protokoll elemzését végezték el.

Az FDT és SPEX nyelvi javaslatok jelen munka kiinduló pontját képezték. Konstrukcióik közül sokat (nem mindig eredeti formájukban) átvéve, a nyitott problémák tisztázásával, uj elemek hozzáadásával alakitottuk ki a [Kovács 82b] dolgo-

zatban előzetesen publikált nyelvi javaslatot. Ugyancsak beszámoltunk a [Kovács 82b] közleményben a jelen munkában részletesen ismertetett félautomatikus protokoll verifikáló rendszer megvalósitásáról is.

2.3 Egyéb módszerek

Ebben a részben olyan protokoll specifikációs és verifikációs módszerekkel fogunk foglalkozni, amelyek nehezen lennének beilleszthetők az előző fejezetek rendszerébe. Két témakör köré csoportositjuk mondanivalónkat, a temporális logikát illetve a formális nyelveket alkalmazó leiró módszereket tárgyaljuk.

2.3.1 Temporális logikai leirás

Az utóbbi években nagy érdeklődés kiséri azokat a kisérleteket, amelyek temporális logikai formalizmust próbálnak alkalmazni specifikációs célokra, a hardver specifikációtól egészen a protokollok területén történő felhasználásig. Az eddigi tapasztalatok alapján a meglévő specifikációs módszerek mindegyike rendelkezik olyan sajátsággal, amely megnehezíti bizonyos körülmények között a használatukat. Véges állapotu gépeket alkalmazó bonyolult rendszerek esetén az állapotok hallatlanul nagy száma sokszor kérdésessé teszi a verifikálhatóságot, például a protokoll progress tulajdonságának bizonyitását. A legtöbb modell absztrakt implementációnak tekinthető, amely sok esetben feleslegesen korlátozza a valódi implementáció készitőit, mert egyes részei, megoldásai magából a leirásból következnek, nem hordoznak lényeges tervezői gondolatokat. A temporális logikai formalizmussal a tervező a protokoll lényeges tulajdonságaira koncentrálhat. A bemutatott példák nem tekinthetők még kiforrott módszereknek, sokkal inkább kezdeti kisérleteknek.

A [Schwartz és m. 81] közlemény a Bitalternáló protokoll temporális logikai specifikációjával foglalkozik. Az elosztott protokoll rendszert egy processzoron "futó" multiprocess rendszerrel modellezi. Ezen háttér modellre irja fel a protokolltól elvárt tulajdonságok logikai kifejezéseit (axiómákat). A kifejezésekben az alapvető temporális logikai operátorok $\Diamond$ "lehetséges", $\Box$ "szükségképpen" mellett a primitiv operátorokból felépithető bonyolultabb operátorokat is alkalmaz.

Jelölje t az időt, i pedig a jelenlegi pillanatot. A P predikátumra alkalmazott "box" operátor jelentése:

$$\Box P \equiv \forall t > i \quad P(t)$$

a P igaz értékü a jelen és minden további pillanatban. A duális operátor a "diamond":

$$\Diamond P \equiv \neg \Box \neg P \equiv \exists t > i \quad P(t)$$

vagyis a jövőben lesz olyan pillanat, amikor P igazzá válik. A formalizmus nem követeli meg a rendszer állapot explicit kifejtését, bár a rendszer modell egy (Q,R) halmazpár, amelyben a Q a rendszer állapotainak halmaza, mig R egy reflexiv és tranzitiv reláció Q-n. A bináris UNTIL operátor jelentése:

$$P \text{ UNTIL } Q \equiv \forall t \geq i \; (\forall i \leq t' \leq t \quad \neg Q(t')) \supset P(t)$$

A Q vagy igaz a jelen pillantban, vagy a P igaz addig amig a Q igazzá nem válik. Az UNTIL operátor segitségével kifejezhető a

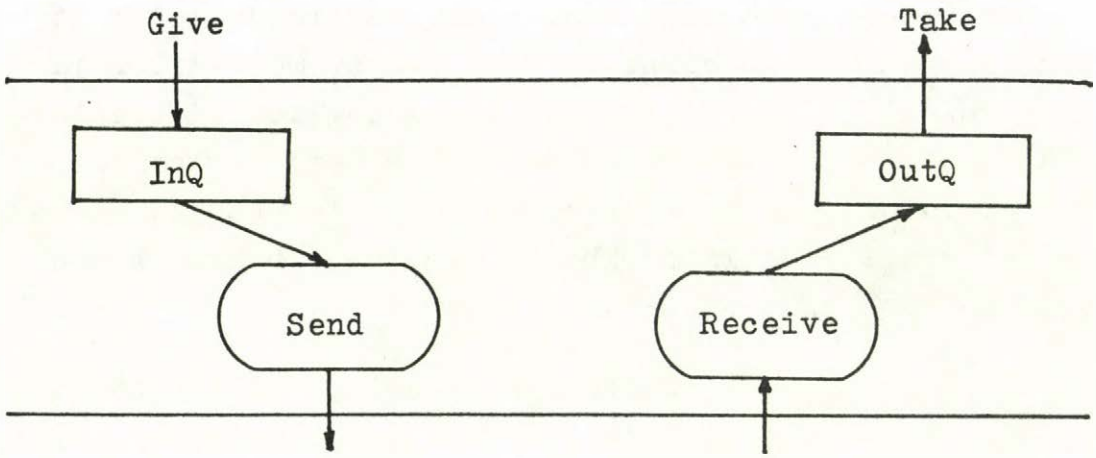
$$P \text{ UNTIL-AFTER } Q \equiv \quad P \text{ UNTIL } (P \wedge Q)$$

$$P \text{ LATCHES-UNTIL } Q \equiv (P \supset (P \text{ UNTIL } Q)) \text{ UNTIL } Q$$

$$P \text{ LATCHES-UNTIL-AFTER } Q \equiv \quad P \text{ LATCHES-UNTIL } (P \wedge Q)$$

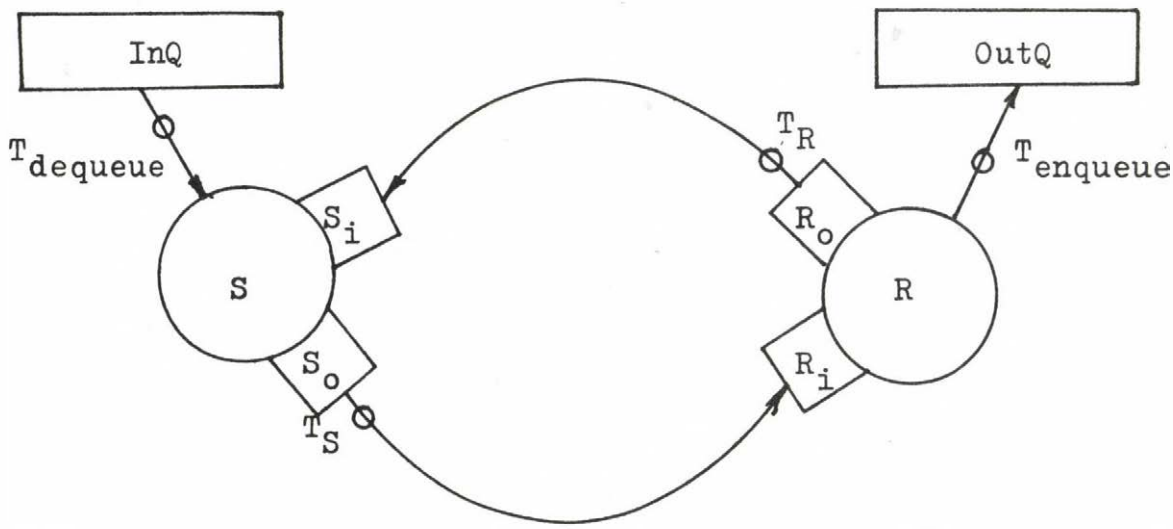
bonyolultabb operátorok. A leirásban az AT, IN, AFTER predikátumok a paralell processek vezérlési (control) pontjainak elhelyezkedésére utalnak. Az operátorok és vezérlési predikátumok felhasználása nem rögziti az adott tulajdonság eléréséhez szükséges állapotátmenet sorozatot, igy szükségtelenül nem korlátozza az implementációt. A Bitalternáló protokoll specifikációjának vázlata látható a 2.12 ábrán. A protokoll modell két processből és a hozzájuk kapcsolódó

két üzenet sorból áll.



2.12 ábra A protokoll specifikáció strukturája

A részletesebb 2.13 ábrán már az S_i, S_o, R_i, R_o input és output puffereket és $T_{dequeue}$, $T_{enqueue}$, T_S, T_R vezérlési pontokat is feltüntettük.



2.13 ábra A protokoll specifikáció részletesebb felépitése

Az üzeneteket az $<m,v>$kifejezéssel jelöljük, ahol m az üzenet, v pedig a sorszám. A specifikációt teljes egészében terjedelmi okokból bemutatni nem tudjuk, illusztrációként két axiómát közlünk.

$$AFTER(T_{dequeue}) \wedge nextval=a \wedge S_o = <m,p> \supset$$
$$S_o = <m,p> \ UNTIL \ S_o = <a,\overline{p}>$$

Jelentése: Az S_o puffer előző értéke $<m,p>$, az InQ sorból az "a" üzenetet kivéve az S_o következő tartalma $<a,\overline{p}>$lesz.

$$S_o = <m,v> \wedge \neg Empty(InQ) \supset$$
$$\left[\Diamond (\neg Corrupted(S_i) \wedge S_i = S_o) \supset \Diamond (S_o \neq <m,v> \wedge AT(T_S))\right]$$

Jelentése: Ha az InQ input sor nem üres akkor egy hibátlanul visszaérkezett nyugta üzenet ujabb üzenet elküldését eredményezi.

Ez a megközelités jelentősen különbözik a [Hailpern és m. 80] dolgozatban közölttől, melyben a rendszer belső struktúrájának előzetes feltételezése nélkül un. história változókra felirt temporális logikai kifejezések adják a rendszer elvárt tulajdonságainak megfogalmazását.

2.3.2 Formális nyelvek

A [Harangozó 78a] közlemény reguláris nyelvtanokat használ protokoll leirásra. A módszer alapja az, hogy a protokoll entitások közötti információ cserét jellemző üzenet sorozatokat egy formális nyelv mondatainak feltételezve a mondatokat nyelvtanokkal generálja. Az entitásokat leiró nyelvtanok előállitására algoritmust közöl.

A mondatszerkezetü (phase-structure) grammatika egy halmaz négyes:

$$G = (V_N, V_T, P, S)$$

ahol

V_N a nemterminálisok,
V_T a terminálisok, $(V = V_N \cup V_T)$
P az átirási szabályok (produkciók) halmaza.
S a kezdő szimbolum $(S \in V_N)$

A produkciók formája $\alpha \longrightarrow \beta$ ahol $\alpha \in V^+$, $\beta \in V^*$ és α-ban van legalább egy nemterminális. A γ szó levezethető az δ-ból ($\delta \overset{*}{\Longrightarrow} \gamma$) véges sok lépés alatt, ha van olyan produkció sorozat, amelyet egymás után alkalmazva δ-ból eljuthatunk γ-ba. A G nyelvtan generálta L(G) nyelv:

$$L(G) = \left\{ w : w \in V_T^* \text{ és } S \overset{*}{\Longrightarrow} w \right\}$$

A protokollt leiró nyelvtan előállitásának algoritmusa a következő:

1.a A V_T terminálisok meghatározása az entitások egymásnak küldött üzenetei alapján.
b A V_N nemterminálisok meghatározása.
c Az átirási szabályok (P) felállitása a protokoll algoritmusa (üzenet sorrendje) alapján.
2. Az üzenetek (framek) belső strukturája alapján a terminálisok dekompoziciója.
3. A (2) pont dekomponált terminálisainak generativ nyelvtannal történő előállitása. A nyelvtan előállitása az 1. pont alapján.
4. Az általános nyelvtan megalkotása a különféle absztrakciós szinteknek megfelelő nyelvtanok helyettesitésével. (A dolgozatban használt reguláris nyelvtanok halmaza zárt a helyettesités müveletére nézve.)

A dolgozatban a HDLC protokoll (normál response mód, fél duplex átvitel) nyelvtannal történő leirását találjuk. A produkciók közlése mátrixos formában történik. A módszer részletes magyar nyelvü elemzése alkotja a [Harangozó 78b] disszertációt.

A [Teng és m. 78a] közleményben az előzőhöz hasonló módszerrel környezet független (context-free) protokoll leiró nyelvtanokat konstruálnak. A szerzők kifejtik, hogy a különböző protokoll tipusok leirására más és más grammatika osztályok alkalmasak, de gyakorlati megfontolásból (elemezhetőség) csak a környezet független nyelvtanokra szoritkoznak. Az ARPANET IMP (Interface Message Processor) két logikailag független részét leiró nyelvtanok megalkotása után a nyelvtanok egyesitését a keverés (shuffle) és a helyettesités (substitution) müveletek segitségével végzik.

A [Teng és m. 78b]-ben a szerzők általánositják korábbi eredményeiket, foglalkoznak az automatikus protokoll implementáció és validáció kérdéseivel is. Az általánositott protokoll modell két nyelvtanból, az üzenet és akció nyelvtanból (message grammar, action grammar) áll. Az üzenet nyelvtan az üzenetek belső szerkezetét, az akció nyelvtan pedig a protokoll előirta müveletek sorrendjét definiálja. A protokoll tervezés lépésenkét finomodó módszerét mutatják be, amelynek utolsó lépése egy szintaxis vezérelt protokoll implementáció előállitása. A protokoll szintaxisát leiró nyelvtanok mellé a szemantikát végrehajtható függvények segitségével definiálják. A protokollt leiró szemantikával kiegészitett nyelvtant a protokoll interpreter (GPS) forditja le önállóan futtatható programmá (parser vezérelt adatstrukturákká). A szerzők utalásokat tesznek a protokoll modell szintaktikai hibáinak és egyes a megvalósitásból eredő korlátozások (balrekurzió mentesség a visszalépés nélküli parsolás miatt) betartásának automatikus ellenőrzésére is.

9. SZÁMITÓGÉPPEL TÁMOGATOTT PROTOKOLL VERIFIKÁLÓ RENDSZER

A következőkben a 8.3 fejezet bővitett állapotelérhetőségi analizisét megvalósitó interaktiv protokoll verifikáló rendszer felépitését, müködését és használatát fogjuk bemutatni. Részletesen elemezni fogjuk azokat a megfontolásokat is, amelyek a megvalósitás különféle megoldásaihoz vezettek.

A protokoll verifikáló rendszer a PDP-11 sorozat kompatibilis számitógépein (LSI-11/03, PDP-11/23, PDP-11/34, PDP-11/40) illetve TPA-11/40-en futó MODULA-2 rendszer [Geissmann 81] alatt iródott. A MODULA-2 rendszer MODULA-2 forditóprogramból, szerkesztőből (linker) és post-mortem nyomkövetőből (debugger) áll. A rendszer RT-11 SJ operációs rendszer alatti futtatását a tárrezidens rendszerkomponens (basic executive) biztositja.

A protokoll verifikáló rendszer MODULA-2 nyelvü modulokból épül fel, amelyek leforditott formában (LNK kiterjesztésü fájlokban) állnak rendelkezésre. A felhasználó ezen fájlokhoz linkeli a protokoll specifikációjának előfeldolgozás illetve forditás utáni formáját. Igy a szerkesztőprogram segitségével egy egységes futtatható forma alakul ki, amely mind a verifikáló rendszert, mind az aktuális protokoll specifikációját magában foglalja. A 9.1 ábra a rendszer használatának menetéről nyujt felvilágositást. Lényegében a 4.1 ábrán látható egységes tervezési módszer konkretizálásának fogható fel. A protokoll specifikációs nyelven (6. fejezet) leirt protokoll definiciót egy forditóprogram MODULA-2 nyelvre forditja. Az ilyen megoldás előnye az, hogy viszonylag kis ráforditással elkészithető a precompiler mig legfőbb hátránya az, hogy problémát jelent a szemantikus ellenőrzések a két forditó közötti megosztása. Mindez a forditó gyakorlati felhasználását jelentősen megnehezitheti. A specifikációs nyelvről MODULA-2-re forditás szabályai azon-

3. A PROTOKOLLOK FORMÁLIS SPECIFIKÁCIÓJÁVAL ÉS VERIFIKÁCIÓJÁVAL SZEMBEN TÁMASZTOTT KÖVETELMÉNYEK

Az alábbiakban a formális protokoll specifikációval és verifikációval szemben támasztott követelményeket soroljuk fel, majd rátérünk arra, hogy ezen követelmények hogyan konvertálhatók protokoll specifikációs nyelvi követelményekké.

3.1 A formális protokoll specifikáció követelményei

A protokoll tervezés (specifikáció) és protokoll megvalósitás (implementáció) térbeli és időbeli elkülönülése napjaink alapvető tendenciája. A tervezőket és implementálókat több ezer kilométeres távolságok, országhatárok választhatják el egymástól. Ilyen körülmények között az implementálóknak legtöbbször nincs lehetőségük a tervezőkkel való közvetlen kapcsolat (együttmüködés) megteremtésére. Az egyetlen kommunikációs lehetőséget a protokoll specifikáció hordozza. A formális protokoll specifikáció célja tehát az ember-ember közötti információ csere megvalósitása.

A protokoll tervező - protokoll felhasználó, protokoll tervező - implementáló közötti információ csere megvalósitásának elengedhetetlen feltétele a protokoll specifikáció könnyü megérthetősége. Ez azt jelenti, hogy a tervező gondolatait korlátozások nélkül tudja kifejezni az adott formalizmuson belül és az igy elkészitett specifikációt a leirás olvasója minél kevesebb szellemi energia mozgósitásával tudja befogadni. Mindez nem kedvez a radikálisan uj definiciós technikák széles körü, gyors elterjedésének. Az a specifikációs módszer számithat az implementálók bizalmára, amely jól alkalmazkodik ezen emberek ismeret, tudás, tapasztalat rendszeréhez.

A protokoll definiciójának teljesnek kell lennie. A teljesség azt jelenti, hogy nem maradhatnak nem specifikált

részek a formális leirásban. A hiányokat ugyanis az implementálóknak mindenképpen ki kell tölteni, sok esetben anélkül, hogy valójában maguk tudnának róla. Ennek eredménye inkompatibilitás, inkonzisztens értelmezés lehet.

Hasonló problémákhoz vezet a protokoll specifikáció többértelmüsége. A természetes nyelvi leirással szemben a formális leirás egyik célja a többértelmüségek kiküszöbölése, a specifikáció egyértelmüségének biztositása. Az egyértelmüséget a leirás szintjén kell értelmezni, tehát a leirás nem korlátozhatja szükségtelenül az implementációs lehetőségeket.

Kivánatos, hogy a leirási módszer eszközöket nyujtson a protokoll alapvető strukturáinak (réteg, modul) tömör definiálásához. Célszerü, hogyha a megközelitési mód maga is réteges felépitésü, az egyszerü, átfogó, absztraktabb leirástól a részletesebb felé halad. Az ilyen hierarchikus megközelités nemcsak a megértésben, hanem az implementációban is nagy segitséget tud nyujtani.

Végül, de nem utolsó sorban a protokoll leirása alapján valamilyen formális protokoll helyesség ellenőrző (verifikációs) eljárás alkalmazhatósága igen fontos követelmény. A verifikációs eljárást közvetlenül az eredeti definicióra kell tudni alkalmazni, ugyanis minden köztes transzformáció hiba lehetőséget hordoz magában.

A természetes nyelvi leirás hátrányait korábban már elemeztük, de egy nagyon absztrakt leirás mellett a használata nélkülözhetetlen.

3.2 A formális protokoll verifikáció követelményei

A protokoll tervezés elengedhetetlen lépése annak a vizsgálata, hogy a protokoll megfelel-e a vele szemben támasztott elvárásoknak. Az ilyen tipusu formálisan elvég-

zett vizsgálatokat protokoll verifikációnak nevezik. Az irodalomban megkülönböztetik a protokoll verifikációt a protokoll validációtól. A validáció tágabb jelentéskörü, a verifikáción kivül a tesztelést, szimulációt stb. is magában foglalja. Jelen dolgozatban protokoll verifikáció alatt a protokollok helyességének bizonyitását, olyan formális módszereket értünk, amelyek figyelembe veszik a rendszer müködésének összes lehetséges esetét.

A protokollal szemben támasztott legfontosabb elvárás az, hogy a protokoll valamilyen kommunkációs szolgáltatást nyujtson. Ez alapján jól elkülönithető verifikációs feladat tipust alkot a protokoll nyujtotta és a tőle elvárt szolgátatások összehasonlitása. A protokoll szolgáltatások absztrakt leirására a szolgáltatás specifikáció hivatott. Az eredetileg egységes protokoll specifikáció két részre, szolgáltatás és protokoll specifikációra bontása összhangban van a specifikációval szemben támasztott réteges felépités követelményével. A protokoll felhasználóját csak a protokoll nyujtotta kommunikációs szolgáltatások érdeklik, az hogy milyen szolgáltatásokat várhat a protokoll alkalmazásától és az, hogy ezeket a szolgáltatásokat hogyan (milyen formátumban) lehet igénybe venni. A felhasználó szempontjából a protokoll egy "fekete doboz", amelyet input-output müködése egyértelmüen leir. Ez a verifikációs feladat tipus tehát azt vizsgálja, hogy kielégiti-e a protokoll a szolgáltatás specifikációját vagy sem.

Ezzel kapcsolatban azonban felmerül a protokoll verifikáció alapproblémája. A probléma abból származik, hogy egy protokoll sohasem önmagában nyujtja a tőle elvárt kommunikációs szolgáltatásokat, hanem csak "többlet értéket" (többlet szolgáltatást) ad az alatta fekvő protokoll réteg(ek) szolgáltatásaihoz. Ebben a megvilágitásban tehát már nincs értelme egy adott protokoll helyességéről és ennek megfelelően verifikációjáról beszélni. Az n-edik szinten elhelyezkedő protokoll szolgáltatás specifikációja

az (n-1)-edik, ... 1. szint egymásra rakódott szolgáltatásait is tartalmazza. Egy n-edik szintü verifikációs vizsgálat csak azt tudja kimutatni, hogy a protokoll rendszer az n-edik szinten valóban megfelel-e az n-edik szint szolgáltatás specifikációjának, de az esetleges hiba helyét (azt, hogy a hiba melyik réteg nem megfelelő müködéséből származik) nem. Rétegenkénti protokoll verifikáció arra a feltételezésre épül, hogy az alacsonyabb rétegek protokolljai helyesek. N-edik szintü protokoll verifikációnak előfeltétele az n-1-edik réteg helyes müködése. Ilyen módon egy bottom-up jellegü verifikáció sorozattal a számitógép-hálózat teljes protokoll rendszerének helyessége (elméletileg) belátható. Mindez rávilágit a formális protokoll specifikációval szemben támasztott korábban még nem tárgyalt követelményre, arra, hogy a specifikációnak az alacsonyabb rétegek müködését is tartalmaznia kell (legalábbis szolgáltatás specifikáció szinten). Megjegyezzük, hogy az (n-1)-edik szintü szolgáltatás specifikáció nem azonos az n- és (n-1)-edik szintek közötti interfész leirásával. Az interfész specifikációja a két réteg között "áramló" jelek leirását tartalmazza, mig az (n-1)-edik szint szolgáltatás specifikációja ennél többet, a müködés absztrakt leirását nyujtja. Egy n-edik szintü protokoll formális verifikációjának előfeltétele tehát az (n-1)-edik szint szolgáltatás, az n- és (n-1)-edik szintek közötti interfész, az n-edik szint protokoll és szolgáltatás specifikációjának megléte.

Alapvető elvárás az, hogy a protokoll verifikációs módszer tegye lehetővé a protokoll általános tulajdonságainak (deadlock mentesség, progress tulajdonság stb.) felderitését. A verifikációs módszereket ebből a szempontból két fő csoportba sorolhatjuk. A "globális szemléletü" módszerek a rendszer teljes, minden részletre kiterjedő állapot információja alapján müködnek. Tipikus példa erre az állapotelérhetőségi analizis. Ezzel szemben a "lokális szemléletü" módszerek a protokoll rendszer adott entitását és an-

nak közvetlen rendszerbeli környezetét használják a fenti tulajdonságok meghatározására. A programhelyesség bizonyitáson alapuló verifikációs módszerek ilyen "lokális szemléletüek".

Harmadik tipusu protokoll verifikációs osztályt alkot a protokoll implementáció verifikációja. Ez esetben az elkészült berendezéseket (hardver és szoftver egységeket) vallatják abból a szempontból, hogy megfelelnek-e a protokoll specifikációjában lefektetett tervezői elképzeléseknek vagy sem. Az implementáció vizsgálata a legösszetettebb, legbonyolultabb verifikációs feladat. Nehézsége abból fakad, hogy az implementáció valódi paralellitásokat tartalmazó elosztott hardver, szoftver rendszer. Megfigyeléséhez bonyolult, nagy inteligenciáju müszerek, mérő automaták szükségesek, a mérések pedig sajátos méréstechnikai problémákat (etalon, real-time mérés problémája) vetnek fel. Jelen dolgozatban ezen verifikációs feladat tipust kirekesztjük vizsgálódásunk hatóköréből.

3.3 A protokoll specifikációs nyelvvel szemben támasztott követelmények

Dolgozatunkban protokollok formális leirására specifikációs nyelvi javaslatot adunk. A specifikációs módszerek 3.1 fejezetben felsorolt általános követelményeiből kiindulva az "ideális" protokoll specifikációs nyelvvel szemben támasztott követelményeket elemezzük. Felsoroljuk azokat a nézetünk szerint legfontosabb tulajdonságokat, amellyel a jó specifikációs nyelvnek rendelkeznie kell. Ezen tulajdonságok nagy része megegyezik a modern magasszintü programozási nyelvek elvárt tulajdonságaival, mig mások utalnak a nyelv speciális felhasználási területére.

A protokoll specifikáció megérthetőségének nézetünk szerint egyik legfontosabb komponense a specifikáció olvashatósága. A természetes nyelvnek mindenki birtokában van,

igy nem meglepő az, ha megkivánjuk, hogy a mesterséges nyelv álljon közel a természetes nyelvhez. Az olvashatóságot jelentősen megkönnyiti, hogyha a specifikációs nyelv kulcsszavaiból és az alkalmasan választott azonositókból értelmes mondatok állithetók össze. Az ilyen "öndokumentáló" jellegü programozási nyelvek biztositják azt, hogy maga a program szöveg tartalmazza a dokumentáláshoz szükséges fontosabb információkat, kiküszöbölve ezzel a gyakori kommentálás szükségességét.

A protokoll rendszerek nagy mérete és nagy bonyolultsága miatt alapvető követelmény az, hogy a specifikációs nyelv nyelvi eszközökkel támogassa a részenkénti (modul, szint) protokoll tervezést azt, hogy a modulokat, szinteket egymástól nagymértékben függetlenül lehessen megtervezni és az esetleges változtatások lehetőleg ne követeljék meg a teljes protokoll rendszer strukturájának módositását.

A korábban már emlitett megérthetőség fontos tényezője a modulok mérete. Célszerü, hogyha a rendszer specifikációja olyan, az egy gépelt oldalt meg nem haladó terjedelmü részekbő (modulokból) áll, amelyet az olvasó egy szempillantás alatt felfoghat. Az esetleg több száz modulból (oldalból) álló specifikáció végigolvasása után az olvasónak képesnek kell lennie arra, hogy megértse az egész rendszer müködését anélkül, hogy a részletekre visszaemlékezne. Ez csak akkor biztositható, ha a modulok egyszerü, könnyen érthető, minimális interfészen keresztül kapcsolódnak egymáshoz. A specifikációs nyelvnek meg kell oldania a modul interfészen keresztüli információ áramlás vezérlésének problémáját is. Egy modul belső (!) müködése nem függhet más modulok hibás vagy hibátlan lefutásától. A belső müködést kizárólag csak a jól definiált interfészen keresztül kapott adatok határozhatják meg.

A protokoll specifikáció gépi feldolgozhatóságának követelménye azt jelenti. hogy a számitógép a protokoll speci-

fikáció elemzésével képes legyen kimutatni a specifikáció formai (szintaktikai) és (statikus) szemantikai hibáit. Manapság már egyre többen vetik fel a protokoll specifikáció gépi feldolgozhatóságának követelménye mellett a gépi végrehajthatóság követelményét is. A végrehajtás lehet szimbolikus vagy valóságos. Utóbbi esetben azonban nem a hatékonyság, hanem a nyomonkövethetőség (traceability) a döntő szempont. Valószinüleg jelentős változásokat fog hozni az implementáció közeli specifikációs nyelvek szélesebb körü elterjedése. Ez esetben a protokoll specifikációt bizonyos segéd elemekkel kiegészitve és egy kódgenerálási fázist közbeiktatva lényegében kész müködőképes protokoll implementációt kaphatunk.

A dekompozició mellett az absztrakciók alkalmazása az a másik módszer, amellyel jelentősen csökkenthetjük a protokoll tervezés során fellépő problémák bonyolultságát. Az absztrakciók alkalmazása ugyanis nem más mint az adott pillanatban lényeges és lényegtelen részek, tevékenységek szétválasztása. Ilyen módon tehát a protokoll tervező mindig a fontos, a tervezés adott szintjének leglényegesebb kérdéseire koncentrálhat. A konvencionális programozási nyelvek a különféle tipus definiálási, eljárás, szubrutin hivási mechanizmusok bevezetésével próbálják megoldani ezt a problémát. A protokoll specifikációs nyelveknek erősen támogatniuk kell a magasabb szintü objektum és tevékenység absztrakciók alkalmazását. A későbbiekben ismertetett specifikációs nyelv absztrakt adattipusok bevezetésével próbálja megkönnyiteni az absztrakciós lépések megtételét.

A specifikációs nyelvbe épitett nemdeterminizmus ugyancsak hozzájárulhat a tervezés során fellépő rendszerek komplexitásának redukálásához azzal, hogy segitségével az adott pillanatban érdektelen tervezési döntések meghozatala késleltethető.

Az előzőekben felsorolt követelmények alapján nézetünk

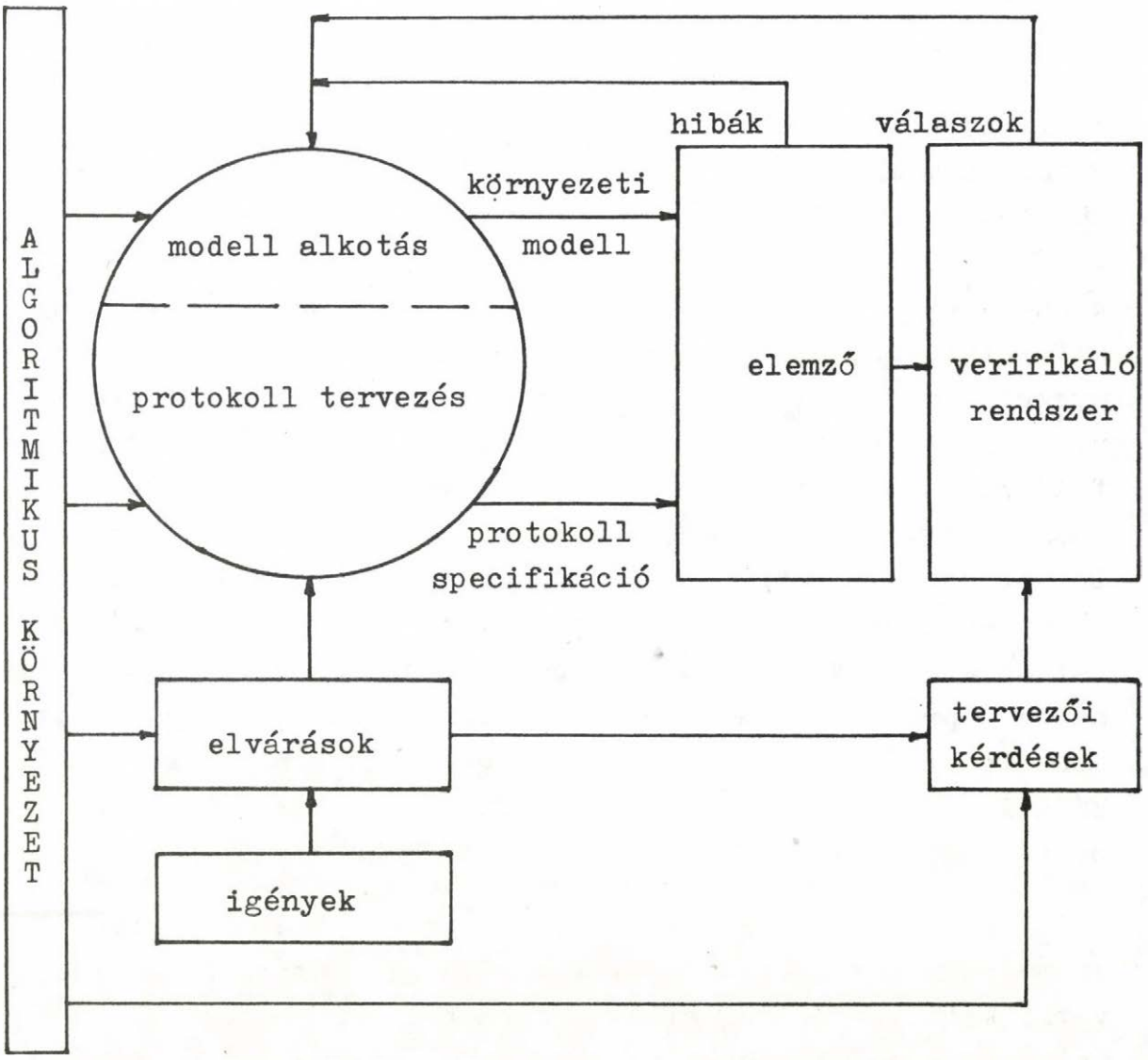
szerint az "ideális" protokoll specifikációs nyelvnek egy

magasszintü,
strukturált,
moduláris felépitésü,
kellő mértékben redundáns,
"öndokumentáló" jellegü,
konkurrens,
nemdeterminisztikus

nyelvnek kell lennie, amelyik erőteljes tipus és egyéb konzisztencia ellenőrzéssel még "forditási időben" ki tudja mutatni a hibák jelentős részét.

4. A PROTOKOLL TERVEZÉSRE JAVASOLT EGYSÉGES MÓDSZER

Az alábbiakban a magasszintü protokoll specifikációs nyelvvel történő protokoll tervezés főbb lépéseire teszünk javaslatot. A 4.1 ábrán vázlatosan feltüntettük a tervezés menetét és az azt támogató szoftver eszközöket.



4.1 ábra Protokoll tervezés specifikációs nyelvvel

A tervező a társadalmi, technikai igényeket felmérve az algoritmikus környezetet figyelembe véve egy követelmény rendszert állit össze, amely a protokollal szemben támasztott mennyiségi és minőségi elvárásokat tartalmazza. A protokoll tervezés végső célja olyan protokoll előállitása, amely maximális mértékben megfelel ennek a követelmény rendszernek. Ezután a környezet és az elvárások alapján a tervező elkésziti a protokoll specifikációját és vele párhuzamosan (!) a protokoll algoritmikus környezetének modelljét is. Az algoritmikus környezet fogalma alatt azokat a protokoll, hardver, szoftver elemeket értjük, amelyek müködése (vagy nem müködése) befolyásolhatja a tervezendő protokoll müködését. N-edik szintü protokoll környezetét az alatta és felette lévő protokoll rétegek képezik. Jelentősen megkönnyitheti a tervező dolgát az, ha ugyanazon nyelven tudja kifejezni mind a protokoll specifikációt, mind pedig a környezeti modellt. (A későbbiekben ismertetett specifikációs nyelv lehetőséget teremt erre.) Az elemző (forditóprogram) felderiti a specifikáció illetve a környezeti modell statikus hibáit. A hibák kijavitása után (amihez a fenti müveletsor esetleg többszöri végrehajtása szükséges) a verifikáló rendszer szoftver támogatásával a tervező választ kaphat az elvárások alapján megfogalmazott kérdéseire, kimutathatja a protokoll rejtett hibáit (deadlock, livelock stb.). Ezen információk ismeretében tovább lehet finomitani illetve kijavitani a protokoll specifikációját és esetlegesen a környezeti modellt is.

A fentiekben ismertetett egységes protokoll tervezési módszer megfelelő szoftver támogatás (forditóprogram, verifikáló rendszer) mellett nagymértékben hozzájárulhat a maiaknál megbizhatóbb kommunkikációs protokollok gyorsabb tervezéséhez, automatikus protokoll implementáció problémáinak jobb megértéséhez. A következő fejezetekben az egyes tervezési lépéseket vesszük alaposabban szemügyre.

5. SZÁMITÓGÉP-HALÓZATI PROTOKOLLOK MATEMATIKAI MODELLJE

A fejezetben fokozatosan "felépitjük" a protokollok, protokoll rendszerek absztrakt elméleti modelljét. A modell eléggé általános ahhoz, hogy ne csak a protokollok, hanem általában az elosztott, párhuzamos digitális rendszerek müködésének formális megfogalmazását is lehetővé tegye. Bizonyos filozófiai általánositások is elképzelhetők, amelyek azonban nem képezik tárgyát jelen dolgozatunknak. A következő fejezetben ismertetett protokoll specifikációs nyelv ezen általános matematikai modell egy lehetséges programozási nyelvi reprezentánsának tekinthető. A modell bevezetésének alapvető célja a specifikációs nyelv matematikai megalapozásán tul, a különféle protokoll tulajdonságok formális megfogalmazása, amelyet a későbbiekben meg is kisérelünk.

Kiinduló pontunk az a plauzibilis definició, amely a párhuzamos számitási rendszereket állapotátmenet rendszerrel azonositja (modellezi). Az állapot fogalma kulcsfontosságu szerepet játszik a rendszerek (és nem csak a digitális rendszerek) leirásában. Alapfogalomnak tekintjük, igy ezen a szinten formálisan nem definiáljuk. Intuitiv definicióként azonban elfogadjuk [Timothy és m. 68] megállapitását: az állapot olyan "információ koncentrátum", amely jellemzi a rendszer multbeli viselkedését és lehetővé teszi a jövőbeli viselkedés kiszámitását.

Definició 5.1 (Keller): Az <u>állapotátmenet rendszer</u> egy (Q,R) halmazpár, melyben a Q a rendszer állapotainak halmaza, R pedig a Q halmazon értelmezett bináris reláció. □

Ha a q, q'$\in$ Q állapotok R relációban állnak egymással, akkor azt a q$\longrightarrow$q' kifejezéssel fogjuk jelölni és állapotátmenetnek (a későbbiekben egyszerüen csak átmenetnek) nevezzük. A Q és R halmazoktól nem követeljük meg a végességet, de a valóságos (realizálható) rendszerek esetén Q és R szük-

ségképpen véges[5], igy a későbbiekben erre szoritkozunk. A $q, q' \in Q$; $q \longrightarrow q'$ állapotátmenet során a rendszer (ezen a szinten) oszthatatlan, atomi müveletet (akciót) hajt végre, amelyet állapot aktualizálásnak nevezünk.

Definició 5.2 : A (Q,R) állapotátmenet rendszer egy $q \in Q$ állapotában a potenciálisan lehetséges következő állapotokat az

$$E(q) = \{ q' : q \longrightarrow q' \}$$

halmaz adja meg. □

Definició 5.3 : Adott (Q,R) és $q \in Q$ esetén a potenciálisan lehetséges állapotátmenetekhez tartozó akciókat végrehajtásra engedélyezett (vagy egyszerüen csak végrehajtható) akcióknak nevezzük. □

Az állapotátmenetek során végzett akciókra (müveletekre) való egyszerübb hivatkozás kedvéért az akcióknak nevet adunk. A névvel ellátott akciók lehetővé teszik az állapotátmenet rendszer kivülről történő megfigyelését. Feltételezzük ugyanis a rendszer állapotainak külső hozzáférhetetlenségét, rejtett voltát.

Definició 5.4 : A névvel ellátott állapotátmenet rendszer egy (Q,R,N) hármas, ahol (Q,R) az 5.1 definició állapotátmenet rendszere, N pedig az akciók nevei. Az egyszerüség kedvéért feltételezzük, hogy az akciók és az N nevek között egy-egy értelmü megfeleltetés létesithető . □

A (Q,R,N) rendszerbeli állapotátmeneteket $q \xrightarrow{n} q'$ formalizmussal jelöljük, ahol $q, q' \in Q$ és $n \in N$ az állapotátmenet során végzett müvelet neve. A későbbiekben (állapot) átmenet rendszer alatt mindig névvel ellátott átmenet rendszert értünk.

Definició 5.5 : A (Q,R,N) átmenet rendszer determinisztikus ha $\forall q \in Q$-ra $\#E(q) \leqq 1$. (A " $\#$ " jel a halmaz

számosságát, véges halmazok esetén az elemek számát jelöli.)
□

Definició 5.6 : A (Q,R,N) átmenet rendszer nemdeterminisztikus szekvenciális ha $\exists q \in Q$, hogy $\#E(q) > 1$. □

A definicióban lévő szekvenciális jelző arra utal, hogy bár lehetséges hogy egy $q \in Q$ állapotban több végrehajtható akció van, azonban azok közül csak egy fog ténylegesen végrehajtódni, egyetlen potenciálisan lehetséges következő állapot fog aktualizálódni. Az átmenet rendszer müködését tehát állapotsorozattal adhatjuk meg.

Definició 5.7 : A (Q,R,N) átmenet rendszer müködése egy

$$q_0, q_1, q_2, \cdots q_i, \cdots$$

állapotsorozat, amelyre teljesülnek a

$\forall i$-re $q_i \in Q$,
q_0 a kezdeti állapot és
$\forall i > 0$-ra $q_i \in E(q_{i-1})$

feltételek. □

A rendszer állapotainak nem szükségszerü külső megfigyelhetősége miatt az 5.7 definicióval ekvivalens rendszer müködés definiciót kapunk ha azt az akciók sorozatával irjuk le.

Definició 5.8 : A (Q,R,N) rendszer $q \in Q$ állapotához tartozó végrehajtható akciókat a

$$V(q) = \left\{ n : n \in N;\ q' \in E(q);\ q \xrightarrow{n} q' \right\}$$

jelöli. □

Definició 5.9 : A (Q,R,N) átmenet rendszer müködése egy

$$n_1, n_2, \cdots n_i, \cdots$$

akció sorozat, melyre $\forall i\ n_i \in V(q_{i-1})$. □

A protokoll rendszerek elméleti modelljeként az 5.6 definició nemdeterminisztikus szekvenciális állapotátmenet rendszerét szeretnénk felhasználni (természetesen további módositások után). Mindehhez a következő alapfeltevéseket kell elfogadnunk.

Alapfeltevés 5.1 : A protokoll rendszereket csak diszkrét időpontokban vizsgáljuk. Ezen időpontokban a rendszert állapota jellemzi. A protokoll rendszer müködése ezen időpontokban felvett állapotok sorozatával megadható. □

Ha a vizsgálat időpontjai t_1, t_2, ... t_i, ..., akkor alapfeltevésünk szerint a

$$q_{t_1}, q_{t_2}, \cdots q_{t_i}, \cdots$$

állapotsorozat egyértelmüen jellemzi a protokoll viselkedését.

Alapfeltevés 5.2 : A diszkrét t_1, t_2, ... t_i, ... időpontokon kivüli pillanatokban a protokoll rendszer állapota nem definiált ("elkent"). □

Leirásunk ilyen módon erősen "szemcsés" szerkezetü, csak diszkrét időpontokban tud biztos információt nyujtani. Két ilyen időpont között (t_i, t_{i+1}) a rendszer müködéséről csak azt tudja megállapitani, hogy valamelyik akció éppen zajlik, de állapotinformációval, annak "elkent" volta miatt nem rendelkezik.

A protokollok párhuzamos számitási rendszerek. Ez azt jelenti, hogy a rendszerben vannak egymással párhuzamosan, átlapolódva végrehajtódó müveletek, akciók. A párhuzamos rendszerek leirására mi a nemdeterminisztikus szekvenciális állapotátmenet rendszereket alkalmazzuk, tehát a paralell rendszereket ilyen rendszerekké transzformáljuk. A transzformációt többféleképpen is elvégezhetjük.

Az első transzformációs módszer a párhuzamosan végzett müveletek (akciók) a priori ismeretére épül. Ez esetben a

rendszer valójában szekvenciális, a lászólagos párhuzamosságok leirása egyszerü akció bővitéssel megoldható. A probléma legtöbbször ugy jelentkezik , hogy a "paralell" (Q,R,N) állapotátmenet rendszerben[6] például az n_1, $n_2 \in N$ akciók párhuzamosan hajtódnak végre. Bevezetve az uj $n' = n_1 \| n_2$ akciót, az $N' = N \cup \{n'\}$ bővitéssel (természetesen ennek megfelelően kell bőviteni az R és esetleg a Q halmazt is) a (Q',R',N') már szekvenciális átmenet rendszer lesz.

A gyakorlatban azonban nem rendelkezünk semmiféle bizonyossággal afelől, hogy két akció bizonyosan párhuzamosan fog végrehajtódni. Ha figyelembe vesszük az akciók végrehajtásának véges idejét (az első transzformációs módszer esetén ezt nem tettük meg), akkor a párhuzamos végrehajtás fogalma rosszul definiáltá válik (csak bizonyos arányu átlapolódásról beszélhetnénk), ezért a paralell rendszerek nemdeterminisztikus szekvenciális rendszerekké való transzformációjánál azt a metódust (második módszer) követjük, hogy a párhuzamosan végrehajtható akciókat (müveleteket) a párhuzamos akciók tetszőleges sorrendü (!) egymás utáni végrehajtásával reprezentáljuk.

A második transzformációs technika hátrányaként emlithetjük meg azt, hogy nem teszi lehetővé az idő formális rendszer keretén belüli preciz használatát. A pontos idő használat azt jelenti, hogy tudva az egyes akciók végrehajtási idejét, a rendszer müködése (5.7 definició) alapján bármely aktualizálódott állapot keletkezésének ideje kiszámitható. Ellentétben az elsővel a második transzformációs módszer erre legfeljebb csak felső korlátot tud szolgáltatni (mivel nem veszi figyelembe az átlapolódásokat).

[5] A megállapitás nem érvényes a szubatomi rendszerekre.

[6] A "paralell" átmenet rendszert nem definiáljuk. Értelmezése: olyan rendszer, amelyben vannak átlapolódó akciók.

A rendszerek elosztott volta mind ez ideig nem jelent meg formális modellünkben. Az elosztottságon alapvetően térbeli elosztottságot értünk azt, hogy bizonyos akciók a rendszer különböző pontjain mennek végbe. Természetesen ez nem befolyásolja az akciók rendszer állapot aktualizálási funkcióját.

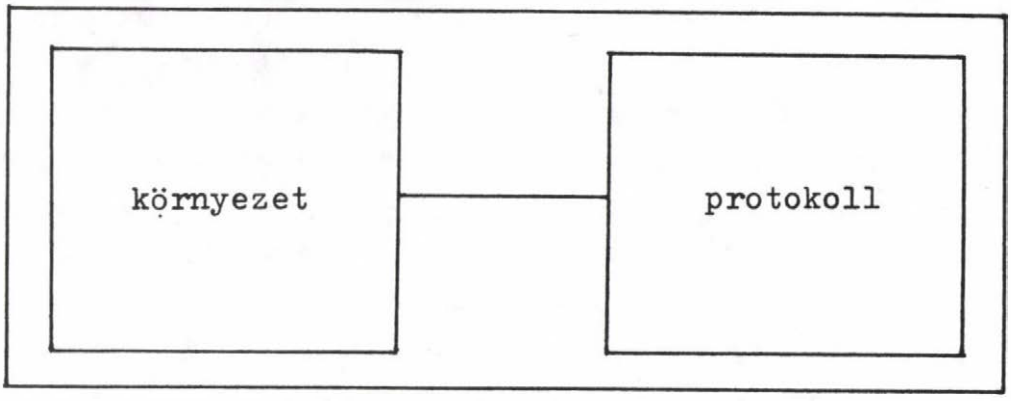
Definició 5.10 : A particionált állapotátmenet rendszer egy (Q,R,N,P) négyes, ahol P az akciók egy felosztása: $p_i \in P$; $p_i \subset N$; $\forall i \neq j$-re $p_i \cap p_j = \emptyset$; $\forall i$-re $p_i \neq \emptyset$. A p_i-ket particióknak nevezzük. □

A későbbiekben csak olyan (Q,R,N,P) átmenet rendszerekkel fogunk foglalkozni, ahol az ugyanazon partícióba tartozó akciók nem hajthatók végre párhuzamosan. Az 5.4 definició homogén rendszerével szemben az 5.10 definició particiói a protokollok szigoruan szekvenciális (de esetleg nemdeterminisztikus) entitásainak modelljei. A

$$q \xrightarrow{p.n} q'$$

jelölés értelmezése: a $p \in P$ partició n akciója aktualizálja a $q' \in Q$ állapotot.

Az 5.10 definició (Q,R,N,P) átmenet rendszerének egy reprezentánsa lehet a (π,α) pár. A (π,α) értelmezéséhez tekintsük az 5.1 ábrát. Az ábrán a "világot" két részre, a protokoll rendszerre és annak algoritmikus környezetére bontottuk. A (Q,R,N,P) rendszerrel a protokollt irjuk le, de nem hagyhatjuk figyelmen kivül azt a tényt, hogy a protokoll és környezete között dinamikus kapcsolat (információ csere) van. Szigoruan szemlélve tehát valójában a protokollt és környezetét egy rendszernek kellene tekinteni és a korábban bevezetett állapotátmenet rendszerrel ezt a "globális" rendszert kellene modellezni. A gyakorlatban ez keresztülvihetetlen. Meg kell elégednünk azzal, hogy a környezetről, annak állapotáról csak részleges információink vannak. Ezeket a részleges állapotinformációkat a $\hat{q}_{\text{környezet}}$ szimbolummal formalizáljuk.



5.1 ábra A protokoll és algoritmikus környezete

Mindezek alapján definiálni fogjuk a protokollt, a protokoll specifikációját. A "természetes" specifikáció a

Definició 5.11 : Totális protokoll specifikáció, a protokoll (Q,R,N,P) modelljének megadása. □

Az 5.11 definició csak elméleti jelentőségü. A gyakorlatban használt protokollok állapottere (Q) és akcióinak (N) nagy száma miatt más utat kell keresnünk. Ezt az uj utat a (Q,R,N,P) rendszerek különféle reprezentációi jelölik ki számunkra. Egy ilyen reprezentáció lehet a (π,α) pár, amelyben a π egy predikátum függvény, α pedig a Q állapotteren értelmezett átmenet függvény. A (Q,R,N,P) és a (π,α) rendszerek közötti megfeleltetést a következő feltételek biztositják.

Ha a (Q,R,N,P) átmenet rendszerben $q, q' \in Q$; $p \in P$; és $n \in N$; valamint

$$q \xrightarrow{p.n} q'$$

akkor és csakis akkor ha a (π,α) reprezentációban

$$\pi_{p.n}(REP(q, \hat{q}_{környezet})) = \text{igaz} \quad \text{és}$$

$$REP(q', \hat{q}_{környezet}) = \alpha_{p.n}(REP(q, \hat{q}_{környezet}))$$

Természetesen a (π,α) rendszerben is megengedjük a nemdeterminizmus lehetőségét, tehát ha egy adott $(q, \hat{q}_{környezet})$ állapotban több átmenet engedélyező előfeltétele (π) igaz értékü, akkor véletlenszerüen választódik ki a valóban végrehajtott átmenet. (A REP függvény az állapotot reprezentáló objektumra utal.) A kifejezésekben szereplő p.n index jelzi, hogy minden átmenethez külön-külön rendelhető π engedélyező predikátum és α átmenet függvény. A (π,α) reprezentációval történő protokoll megadás azonban éppen abban különbözik a (Q,R,N,P) megadásától, hogy segitségével nagy számu állapotátmenet adható meg ugyanazon π,α függvényekkel.

A továbbiakban felülvizsgáljuk az 5.1 definició kapcsán tett megjegyzésünket, melyben az állapotátmenetek során végrehajtott akciók oszthatatlanságáról, atomi jellegéről szóltunk. Egy adott absztrakciós szintről szemlélt atomi akció egy alacsonyabb absztrakciós szintről szemlélve elveszti oszthatatlan jellegét. Erre példaként álljon itt a számitógép valamely müvelet végző egységének regiszter és kapu szintü szemlélete. Ami regiszter szinten egyetlen müveletnek tekinthető (például regiszter transzfer), az a kapu szintről szemlélve apró állapotváltozások sorozatával jellemezhető. A jelenség formalizálása érdekében bevezetjük a k-szintü állapotátmenet rendszer fogalmát.

Definició 5.12 : <u>K-szintü állapotátmenet rendszer</u> k számu (Q,R,N,P) átmenet rendszerből álló

$$^1(Q,R,N,P),\ ^2(Q,R,N,P),\ \ldots\ ^k(Q,R,N,P)$$

sorozat, ahol az első felső index az absztrakciós szint jelölésére szolgál. A sorozat (i+1)-edik eleme az i-edik elemből (i = 2..k) "finomitás" utján keletkezik. A "finomitás" az i-edik szinten oszthatatlan akciók elemibb akciókra bontását, ennek megfelelően uj akciók és állapotok bevezetését jelenti. Az állapot és akció bővitésnél azonban betartandó a következő feltétel: ha az i-edik szinten

$$ {}^{i}q \xrightarrow{{}^{i}p.{}^{i}n} {}^{i}q', $$

akkor az (i+1)-edik szinten az ${}^{i}q$ állapotnak megfelelő ${}^{i+1}q$ állapotból kiinduló összes (i+1)-edik szintü rendszer müködésnek (5.7 definició) tartalmaznia kell az ${}^{i}q'$-nek megfelelő ${}^{i+1}q'$ állapotot. □

Egyszerübben fogalmazva ez azt jelenti, hogy az i-edik szintü állapotok és akciók "felhasadnak", a rendszer alapmüködése megsértése nélkül. A "felhasadások" célja a rendszer "mikroszkopikus" müködésének egyértelmü leirása. Az i- és (i+1)-edik szintü rendszer müködések közötti összefüggés tehát a következő: ha az ${}^{i}(Q,R,N,P)$ rendszer egy müködése

$$ {}^{i}q_1, {}^{i}q_2, \ldots {}^{i}q_j, \ldots ; \qquad {}^{i}q_j \in {}^{i}Q; $$

akkor ennek ${}^{i+1}(Q,R,N,P)$ rendszerbeli megfelelője egy

$$ \left\{ {}^{i+1}w_{1,1}, {}^{i+1}w_{1,2}, \ldots {}^{i+1}w_{1,m}, \ldots : 1 \in \Delta ; \quad {}^{i+1}w_{1,m} \in {}^{i+1}Q \right\} $$

müködés halmaz, melyben minden egyes müködés elszórtan de sorrendtartóan tartalmazza az ${}^{i}q_j$-nek megfelelő ${}^{i+1}w_{1,m}$ állapotok mindegyikét. (Δ a lehetséges müködések (i+1)-edik szintü halmaza.)

Definició 5.13 : A k-szintü állapotátmenet rendszer elemei ${}^{i}(Q,R,N.P)$; i = 1..k ; egymással <u>erősen ekvivalensek</u>. □

Ez alapján a protokollok ekvivalenciája:

Definició 5.14 : Két, $(\pi,\alpha)_1$ és $(\pi,\alpha)_2$ reprezentációju protokoll akkor és csakis akkor erősen ekvivalens, ha a nekik megfelelő $(Q,R,N,P)_1$ és $(Q,R,N,P)_2$ modell erősen ekvivalens. □

A gyakorlati felhasználhatóság tekintetében az erős ekvivalencia tulságosan szigoru feltétel. Olyan ekvivalencia fogalomra van szükség, amely két protokollt csak akkor

különböztet meg, ha müködésük a protokoll környezetéből szemlélve megkülönböztethető. Tehát két protokollt akkor tartunk ekvivalensnek, ha input/output szempontjából ugyanugy viselkednek.

Definició 5.15 : A $(\pi,\alpha)_1$ és $(\pi,\alpha)_2$ gyengén ekvivalensek ha környzetükre ugyanolyan

1^qkörnyezet' 2^qkörnyezet' ··· i^qkörnyezet' ···

állapot változás sorozatot kényszeritenek rá, tehát input/output ekvivalensek. □

A két ekvivalencia fogalom (5.14 és 5.15 definició) közötti összefüggés további vizsgálatot igényel, bár az triviális, hogy az erős ekvivalencia a gyengébbet implikálja.

6. PROTOKOLLOK LEIRÁSA SPECIFIKÁCIÓS NYELV SEGITSÉGÉVEL

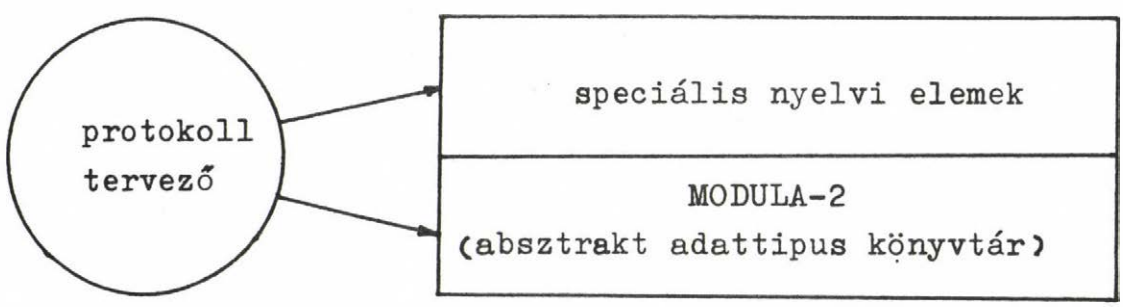
A protokollok elméleti modellje $(\mathcal{T},\alpha)$ reprezentációjának (5. fejezet) egy programozási nyelvi megvalósitására teszünk javaslatot. Az alább ismertetésre kerülő protokoll specifikációs nyelv alapját a MODULA-2 programozási nyelv alkotja. Dolgozatunkban az algoritmusok leirására is ezt a nyelvet használjuk, ezért először röviden vázoljuk alapvető konstrukcióit. Ezután a specifikációs nyelv felépitését mutatjuk be, majd a protokollok leirását lehetővé tévő speciális nyelvi elemeket tárgyaljuk és közöljük a javasolt nyelv egy lehetséges szintaxisát. A nyelv használatára konkrét példa kapcsán térünk ki.

A MODULA-2 a PASCAL és MODULA nyelvek tovább fejlesztett változata, 1980-ban publikálta Wirth [Wirth 80]. Vonzóbb szintaktikai formában megtartva a PASCAL alapelemeit, a MODULA(-2) legfőbb ujitása a modul fogalom bevezetése volt, nem emlitve a párhuzamosság kezelését lehetővé tévő konstrukciókat. Jelen munkában nem használtuk ki a MODULA-2 paralell vonásait, igy erre a következőkben nem térünk ki, helyette a magyar nyelven hozzáférhető [Böszörményi 81] tanulmányra utalunk, amely részletesen elemezi ezeket.

A MODULA-2 magasszintü modul és eljárás orientált erősen tipusos nyelv. Használata nagy segitséget nyujt a strukturált, jól olvasható programok elkészitéséhez. A modul egy "szintaktikus fallal" körülhatárolt zárt egység. Alapvető célja erőteljes hatáskör és láthatóság korlátozás. A modul belsejéből csak azok a külső objektumok érhetők el, amelyeket a modul importál illetve csak azokat tesz a külső felhasználók (modulok) számára hozzáférhetővé, amelyeket exportál. A modulokból felépülő programokban a láthatósági viszonyok tehát gyökeresen eltérnek a megszokott eljárás orientált nyelvekben (pl. ALGOL 60) tapasztalhatókétól. A modul konstans, tipus definiciókat, változó, eljárás, függvény dekla-

rációkat és esetleg kezdeti, végrehajtható utasitásokból álló program részletet tartalmaz. A kezdeti (inicializáló) utasitások a modult tartalmazó eljárás aktivizálásakor végrehajtódnak. A MODULA-2 megengedi a modulok szeparált forditását, ennek érdekében a modult két részre, definiciós és implementációs modulra bontja. A definiciós rész az exportált konstansokból, tipusokból, változókból illetve eljárás és függvény fejekből áll. Az exportált eljárások részletes kifejtése az implementációs modulban helyezkedik el. A definiciós modul tehát a modul a környezete (többi modul) felé mutatott felhasználói interfészét irja le, tartalmazza mindazokat az információkat, amely a modul által nyujtott "szolgáltatások" (pl. eljárások) igénybevételéhez szükségesek. A nyelv változatos tipus definiálási lehetőségeket kinál. Az alaptipusok (INTEGER, CARDINAL, BOOLEAN, CHAR, REAL) mellett a felsorolható (enumerációs), az intervallum (subrange), a tömb (ARRAY) és rekord (RECORD) definiálás is megengedett. A nyelvben megtalálható a halmaz (SET) és pointer tipus. A szokásos operátorokon kivül az IF-THEN-ELSE, CASE elágaztató, a WHILE, REPEAT, FOR, LOOP ciklus utasitások állnak rendelkezésre, nem emlitve az eljárás, függvény definiálási és még számos egyéb lehetőséget.

A protokoll specifikációs nyelvünknek két rétege van 6.1 ábra .



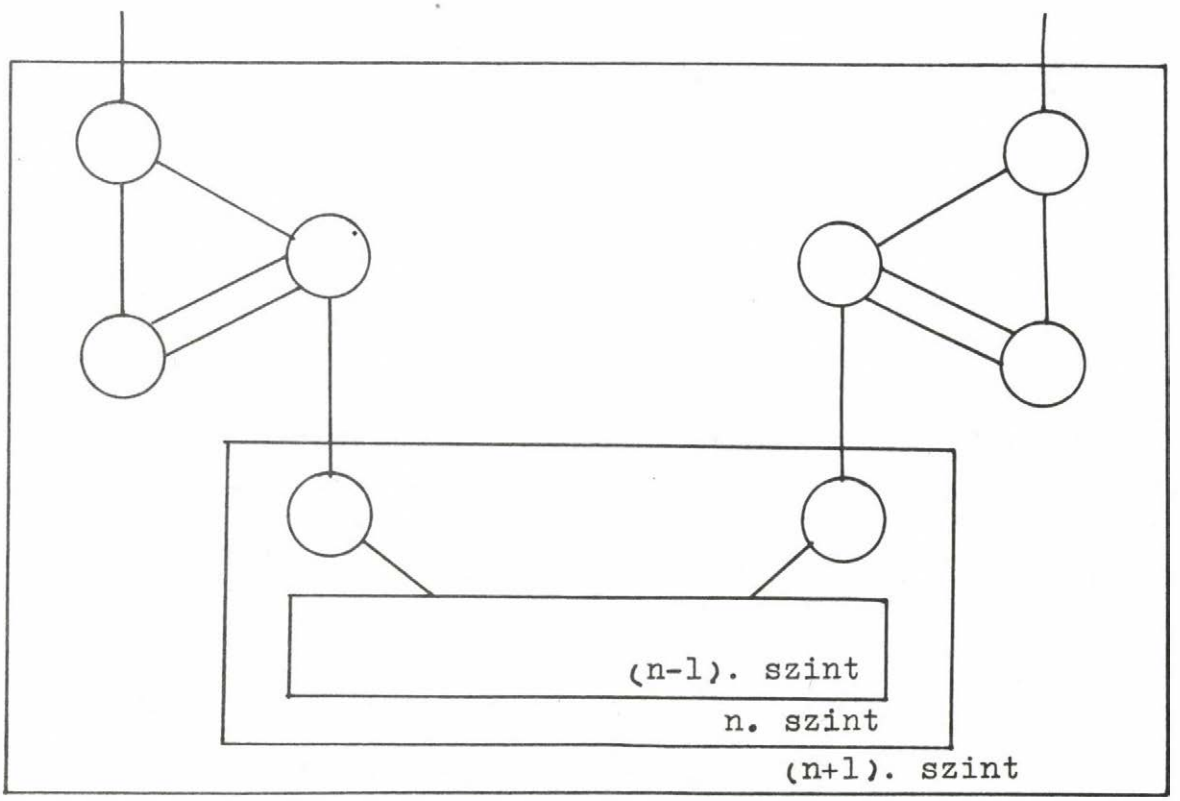
6.1 ábra A specifikációs nyelv felépitése

A nyelv alsó rétegét a MODULA-2 nyelven megfogalmazott, könyvtárba foglalt absztrakt adattipusok alkotják. Az absztrakt adattipusok könyvtárba foglalását a specifikációs nyelvre épülő verifikáló rendszer követeli meg. A bővithető modul könyvtár alapvető célja az, hogy lehetőséget nyujtson a protokollok különféle adatszerkezeteinek és a rajtuk értelmezett müveletek szabad (az általános célu MODULA-2 programozási nyelv strukturáit felhasználó) megfogalmazására. Ez a nyelvi réteg szolgál a bonyolultabb szekvenciális algoritmusok (számitások, adatranszformációk,kódolás, statisztika készités, utképzés stb.) megvalósitására is. A nyelv felső szintjéről az alsó szint mint atomi, oszthatatlan müveletek képviselte szolgáltatás halmaz jelenik meg. Az alsó nyelvi szinttől két alapkövetelmény teljesitését várjuk el. Először is az oszthatatlan müveletektől a véges időn belüli terminálódást, másodszor pedig a robosztusságot kivánjuk meg. A müveletek véges időn belüli terminálódása azt jelenti, hogyha igénybe vesszük egy adott művelet számitási szolgáltatását (meghivjuk az adott müveletet megvalósitó eljárást), akkor a számitás véges időn belül befejeződik (a hivott eljárás véges időn belül visszaadja a vezérlést). A robosztusság alatt az alsó nyelvi réteg azon tulajdonságát értjük, hogy az alsó réteg eljárásai védve vannak a felső réteg okozta minden hibától. A felső réteg az alsó réteg szolgáltatásait eljárás hivások segitségével veszi igénybe. Ezen követelmény teljesitése tehát azt kivánja meg, hogy az eljárások belső müködésében ne okozzon hibát az ha véletlenül nem megfelelő paraméterekkel hivták meg az adott eljárásokat vagy az ha az egymással szoros kapcsolatban lévő eljárásokat (ugyanazon az adatstrukturán értelmezett eljárásokat) nem megfelelő sorrendben használták. A robosztusság biztositásának lehetőségeiről a következőkben még szót ejtünk. A könyvtár bővithetősége ugyancsak fontos elvárás, hiszen az uj protokollokban esetleg teljesen más, a régiektől merőben eltérő adatstrukturák, absztrakt adattipusok fordulhatnak elő.

A specifikációs nyelv felső rétegét a speciálisan protokoll leirásra kialakitott nyelvi elemek alkotják. Ezen nyelvi elemek (rendszer, entitás, környezet) kialakitására legerőteljesebben a SPEX, az FDT javaslatok és a MODULA-2 nyelv bizonyos elemei hatottak.

A specifikációs nyelv a protokollt (protokoll rendszert) egymással és a környezetükkel információt cserélő entitások (ENTITY) halmazának (!) tekinti. Nem tételez fel semmiféle előzetes strukturát, nem korlátozza az entitások közötti kapcsolatok kialakitásának lehetőségét. Ennek megfelelően nemcsak az 1.4 fejezetben már bemutatott réteges, hierarchikus protokoll felépitès leirására alkalmas, hanem az ettől eltérő architekturák specifikálása is könnyüszerrel elvégezhető, bár a réteges felépitést nyelvi eszközökkel is támogatja. Az egymással és környezetükkel kapcsolatban lévő entitások halmazát a nyelv rendszernek (SYSTEM) nevezi. A rendszerben az entitások száma nem korlátozott (ilyen korlátot legfeljebb csak a nyelv implementációja szabhat meg). Összhangban az 1.4 fejezet referencia modelljével, a rendszer entitásai rétegeket alkotnak. Egy rétegbe két vagy több entitás tartozhat. A rétegen belüli entitások vagy közvetlenül (közeli entitások esetén) vagy az alsóbb rétegek kommunikációs szolgáltatásait igénybevéve (távoli entitások esetén) kommunikálnak. Egy ilyen példát tüntettünk fel a 6.2 ábrán. Az ábrán az entitásokat körök, a köztük lévő kapcsolatokat pedig vonalak jelzik. Az (n+1)-edik szinten például három, egymással szorosan összefüggő entitás alkot egy protokoll állomást, mig az n-edik szinten csak egyetlen egy. A szintek közötti kapcsolatok lényegében nem különböznek a többi kapcsolatoktól, ugyanugy két entitás között "feszülnek ki". Mivel a nyelv nem követeli meg a réteg határok kötelező kijelölését (az csak opcionális) és ennek megfelelően nem ellenőrzi azt, hogy egy adott réteg adott entitása melyik másik réteg melyik másik entitásával van kapcsolatban ezért lehetőség van arra, hogy egy entitás az alatta lévő réteget

vagy rétegeket kikerülve szoros kapcsolatban legyen nem közvetlenül szomszédos rétegbeli entitásokkal. Ilyen megoldást sok esetben gazdaságossági, implementációbeli megfontolások indokolhatnak. A kapcsolatok konzisztenciájának entitás szintü ellenőrzéséről a továbbiakban (6.3 fejezet) még szólni fogunk.



6.2 ábra Entitásokból álló hierarchikus rendszer

A protokoll specifikációs nyelv tipikus felhasználási módja lehet az, amikor a tervező egy konkrét réteget ir le. Ez esetben a réteg megtervezése mellett meg kell fogalmaznia azokat az elvárásokat is, amelyeket a tervező az alsóbb szintekkel szemben támaszt. Ki kell tehát alakitani az alsóbb szintek egy formális modelljét is. A nyelv a környezet (ENVIRONMENT) fogalmát használja az ilyen formális modellek megfogalmazására. Egy protokoll réteg környezetébe

az alsóbb szintek modelljén kivül a felső szint bizonyos részei is beletartoznak.

A nyelv lehetőséget teremt arra, hogy verifikációs célból a protokolltól elvárt tulajdonságok, követelmények formálisan is megfogalmazhatók legyenek. Erre szolgál a tulajdonságokat leiró (PROPERTIES) nyelvi elem.

SYSTEM:

entitás leirások	ENTITY
környezet leirása	ENVIRONMENT
kapcsolatok leirása	CONNECTIONS
elvárt követelmények megfogalmazása	PROPERTIES

6.3 ábra A protokoll rendszer specifikációs nyelvi leirása a megfelelő kulcsszavakkal

Összefoglalásképpen tehát egy protokoll rendszer definiálása a komponens entitások formális leirásából, az algoritmikus környezet leirásából és az entitások közötti kapcsolatok leirásából, valamint az elvárt protokoll tulajdonságok megfogalmazásából áll. (6.3 ábra)

6.1 Absztrakt adattipusok

A specifikációs nyelv absztrakt adattipusok bevezetésével próbálja támogatni a protokoll specifikáció absztrakciós lépéseit. Az absztrakt adattipusok fogalmának ismertetése után a tipus specifikáció lehetőségeit elemezzük, majd bemutatjuk a nyelv alsó rétegét képező MODULA-2 alapu adattipus definiciós módszert.

A programozási nyelvek tipus fogalmának alapvetően kétféle megitélése terjedt el. Az első nézet szerint a tipus értékhalmaz, egy adott változó által felvehető értékek halmaza. Az adattipust tehát egy értékhalmaz valamely reprezentációjának megadásával lehet definiálni. Az adattipuson (pontosabban egy adott tipusu változón) végzett müveleteket eljárásoknak tekinti, amelyek azonban függetlenek az adattipustól. A legtöbb programozási nyelv (FORTRAN, PL/1, PASCAL) az adattipus ezen felfogását követi.

A tipus fogalom másik megitélése szerint a tipus nem értékhalmaz, hanem egy olyan nyelvi mechanizmus, amelynek alapvető célja a biztonság fokozása. E nézet szembeállitja a "tipus egyenlő értékhalmaz" felfogást az adattipusok absztrakt, implementáció független felfogásával. Az absztrakt adattipus tehát (absztrakt) objektumok és a rajtuk értelmezett müveletek összefüggő, szétválaszthatatlan együttese. Az objektumokhoz kizárólag csak ezen müveletek utján, csak rajtuk keresztül lehet hozzáférni, csak ezek a müveletek modosithatják az objektumok állapotát (értékét). A programozási nyelvek speciális nyelvi eszközökkel támogathatják az absztrakt adattipusok definiálási lehetőségét, ami azzal az előnnyel járhat, hogy áttekinthetőbb, könnyebben érthető és ezzel szoros összefüggésben megbizhatóbb program termékeket lehet előállitani. Az intenziv tipus ellenőrzés ugyancsak jelentősen hozzájárulhat a programok megbizhatóságának növekedéséhez, mivel nagy számu hiba még forditási időben felfe-

dezhetővé válik. Az első ilyen nyelvi eszköz a SIMULA 67 osztály (CLASS) fogalma volt.

Elméletileg az absztrakt adattipus definiálás szintaktika és szemantika megadásából áll. Az absztrakt adattipus szintaktikája az adattipus nevét, az értelmezett absztrakt operációk (müveletek) nevét, értelmezési tartományát és értékkészletüket irja le legtöbbször felsorolásszerüen. A szemantika definiálására azonban már eltérő módszerek alakultak ki. Az axiomatikus specifikáció Hoare munkáira épül. Az objektumokat absztrakt invariánsok (predikátumok) felhasználásával, a müveleteket pedig elő és utó feltételekkel (pre és post predikátumokkal) adja meg. Az implementációs előirás egy leképezés, amely az absztraktból a konkrét reprezentációra képez le.

Egészen más megközelitést alkalmaz a Guttag féle algebrai szemantika specifikációs módszer [Guttag 80]. A szemantikát axióma halmazzal, algebrai egyenletekkel definiálja. A szerző kidolgozta az ilyen jellegü absztrakt adattipus specifikáció konzisztenciájának (ellentmondás mentességének) és teljességének ellenőrzését lehetővé tévő módszereket is. A 6.4 ábrán példaként bemutatjuk egy T tipusu elemekből álló FIFO sor algebrai specifikációját, amely a szintaxis, a szemantika és bizonyos korlátozások leirásából áll. Az adattipus egy gazdagabb (bővebb) változatának specifikációját később tárgyaljuk.

A fenti szemantika megadásoktól alapjaiban tér el a procedurális (eljárás orientált) megközelités. Lényegében az absztrakt adattipus egy alap (referencia) modell fogalmaival kifejezett implementációjáról, egy alap modellre való visszavezetésről van szó. Az alap modell lehet valamely (célszerüen magasszintü) programozási nyelv vagy a jól ismert VDL (Bécsi Definiciós Nyelv) is. A módszer hallatlan előnye az, hogy a szemantika definiálása a programozáshoz hasonlit, igy nagyon könnyen elsajátitható. Hátrányaként

lehet megemliteni viszont a bonyolultabb müveletek leirásának hosszadalmasságát, ami ebből kövezkezőleg esetleg a megérthetőség rovására megy. Az ilyen leirási mód esetén viszonylag nehezebb lesz a specifikáció ellentmondás mentességének kimutatása. Ugyancsak megnehezül két különböző specifikáció ekvivalenciájának vizsgálata is. A módszer legnagyobb problémája azonban mégis az alap modell szemantikájának kérdése. Az alap modell szemantikáját ugyanis ismét csak valamilyen módon definiálni kell.

```
TYPE Queue [ T:TYPE ]

SYNTAX

  NewQueue                    ==> Queue
  Add       Queue # T         ==> Queue
  Remove    Queue             ==> Queue
  Front     Queue             ==> T
  Empty     Queue             ==> BOOLEAN

SEMANTICS

  DECLARE q:Queue, x:T

  1. Empty(NewQueue()) = TRUE
  2. Empty(Add(q,x))   = FALSE
  3. Front(Add(q,x))   = IF Empty(q)
                         THEN x
                         ELSE Front(q)
  4. Remove(Add(q,x))  = IF Empty(q)
                         THEN NewQueue()
                         ELSE Add(Remove(q),x)

RESTRICTIONS

  Empty(q)   ==> FAILURE(Front,q)
  Empty(q)   ==> FAILURE(Remove,q)
```

6.4 ábra Algebrai absztrakt adattipus specifikáció

Mindezek ellenére a protokollok formális leirásánál mi is a procedurális specifikációs módszert fogjuk használni.

A MODULA-2 nyelv kiváló lehetőséget teremt erre. A modul fogalom felhasználásával elrejtjük az adattipus belső felépitését, a rajta értelmezett müveletek implementációját a külső felhasználók elől. Az adattipushoz való hozzáférést a modul export listája révén szabályozzuk. Az absztrakt adattipust reprezentáló modul exportálja az adattipus nevét (rejtett belső szerkezettel) és az értelmezett absztrakt müveleteket megvalósitó eljárásokat. A modul definiciós része az adattipus szintaktikáját, implementációs része a szemantikáját irja le. A továbbiakban bemutatjuk a 7. fejezet példájában, a Bitalternáló protokoll specifikációjában alkalmazott adattipusok egy csoportjának leirását. Csak szintaktikai leirásra szoritkozhatunk, ugyanis az implementációs részben a csak később ismertetendő protokoll verifikáló rendszer belső felépitésétől függő részek is találhatók.

Az adattipusokon értelmezett müveleteket megvalósitó függvényeket három csoportba sorolhatjuk.

A konstruktorok (constructors) az adattipus értékeinek előállitására szolgálnak. A függvények vagy argumentum nélküliek, vagy értelmezési tartományuk valamely predefiniált alap vagy származtatott tipus(ok). Az értékkészlet a definiálandó adattipus értéktartománya.

A bővitők (extenders) a konstruktorokból származtatható, a konstruktorokra visszavezethető függvények. Az adattipus definiálásához nincs szükség rájuk, de kényelmessebbé tehetik a használatot. Értékkészletük ugyanugy a definiálandó tipus értéktartományába esik.

A szelektorok (selectors) értelmezési tartománya a definiálandó, mig értékkészlete valamely más, eltérő tipus. Feladatuk gyakran a konstruktorok paramétereinek "lekérdezése", vagyis ilyenkor a szelektorok értékkészlete a konstruktorok értelmezési tartományába esik.

A Bitalternáló protokoll csomag formátumának rögzité-

sére szolgál a Packet adattipus.

```
TYPE      Packet;
PROCEDURE MakePacket(m:Message; s:BOOLEAN):Packet;
PROCEDURE Text(pk:Packet):Message;
PROCEDURE SeqNumber(pk:Packet):BOOLEAN;
PROCEDURE EqualPacket(p1,p2:Packet):BOOLEAN;
PROCEDURE NullPacket():Packet;
```

A csomag üzenetrészből (Message) és egy bites sorszámból áll. A sorszám természetes módon BOOLEAN tipusu, amely a MODULA-2 beépitett alaptipusa. A MakePacket konstruktor függvény az üzenetből és sorszámból létrehozza a csomag (Packet) adattipust. Itt jegyezzük meg, hogy az üzenet (Message) tipus specifikálására a Bitalternáló protokoll leirásánál nincs szükség, mivel a protokoll az üzenetek transzparens átvitelére szolgál, tehát nem tételez fel semmilyen előzetes üzenet strukturát. A verifikáló rendszer később megköveteli majd az üzenet tipus valamilyen specifikációját, de jelen pillanatban az üzenetet definiálatlannak foghatjuk fel. A NullPacket bővitő nulla (FALSE) sorszámból és üres üzenetből (NullMessage) állit össze csomagot. A Text és SeqNumber szelektorok hozzáférést engednek a csomag egyes komponenseihez. Az EqualPacket szelektor két szomag összehasonlitásakor igaz (TRUE) értéket ad, ha a két csomag azonos.

A csomagokból álló FIFO sorok kezelését végzi a QueueOfPacket absztrakt adattipus. Az adattipus szintaktikáját alább közöljük. A konstruktorok közül a NewQueueOfPacket egy csomagból álló üres sort hoz létre, mig az Add függvény ujabb csomagot illeszt a sor végére. A bővitők a Remove, a Clear és az Append függvények. A Remove eljávolitja a sor elején álló csomagot, a Clear egymás utáni Remove eljárások alkalmazásával üres sort hoz létre, az

Append pedig két sor konkatenációját valósitja meg, ugy hogy a második sort az első sorhoz füzi. A szelektorok egyik csoportja (Front, Middle, Back) az első, valamely középső és az utolsó csomagok kiválasztására hivatott. Az In predikátum függvény valamely adott csomag, sorban tartózkodásáról, mig az Empty a sor ürességéről nyujt felvilágositást. Az Equal függvény két sor első meghatározott hosszuságu szeletét hasonlitja össze. A Length szelektor a sor aktuális hosszuságát adja meg.

```
TYPE      QueueOfPacket;

PROCEDURE NewQueueOfPacket() : QueueOfPacket;

PROCEDURE Add(VAR q:QueueOfPacket;i:Packet) : QueueOfPacket;

PROCEDURE Remove(VAR q:QueueOfPacket) : QueueOfPacket;

PROCEDURE Append(VAR q1,q2:QueueOfPacket) : QueueOfPacket;

PROCEDURE Clear(VAR q:QueueOfPacket) : QueueOfPacket;

PROCEDURE Front(q:QueueOfPacket) : Packet;

PROCEDURE Middle(q:QueueOfPacket; seq:INTEGER) : Packet;

PROCEDURE Back(q:QueueOfPacket) : Packet;

PROCEDURE In(q:QueueOfPacket;i:Packet) : BOOLEAN;

PROCEDURE Empty(q:QueueOfPacket) : BOOLEAN;

PROCEDURE Equal(q1,q2:QueueOfPacket;slice:INTEGER):BOOLEAN;

PROCEDURE Length(q:QueueOfPacket) : INTEGER;
```

6.2 Az entitás fogalma

Az entitás (entity) mint a protokoll specifikációs nyelv alapfogalma, a környezetétől "szintaktikus fallal" elhatárolt absztrakt objektum. A (Q,R,N,P) elméleti modell particióinak nyelvi reprezentánsa. Belső felépitése a 6.5 ábrán látható.

ENTITY:

állapotváltozók	STATE VARIABLES
lehetséges interakciók	INTERFACES
állapotátmenetek előfeltétel müvelet	TRANSITIONS PRECOND ACTION
kezdeti állapot beállitás	INITIAL STATES

6.5 ábra Az entitás belső felépitése

Az entitás belső állapotterét az állapotváltozói által felvett értékek tartománya határozza meg. Az állapotváltozókhoz csak az entitás belsejéből lehet hozzáférni, tehát egy adott protokoll specifikációjában a különböző entitások azonos nevü állapotváltozói egymástól független változók. Nem követeljük meg az entitás belső állapotainak explicit megnevezését (mint ahogy azt az FDT szorgalmazza) de meg sem tiltjuk. A tervező használhat explicit állapot elnevezést egy felsorolható (enumerációs) tipusu állapotváltozó bevezetésével. Az állapotváltozók számát nem korlátozzuk.

Az entitás a környezetével (és más entitásokkal) kizá-

rólag interakciók utján kommunikál. Az interakció az entitás belsejéből elérhető interfész objektumokon végzett atomi müvelet. A lehetséges interakciók megadásához egyértelmüen meg kell adni az interfészváltozókat és a rajtuk értelmezett müveleteket. Az engedélyezett müveletek megadása implicit módon, az objektumok tipusának megadásával történik. A kommunikáció részletes megvalósitásáról a következő fejezetben fogunk szólni.

Az entitás a környezetéből illetve más entitásoktól kapott interakciók hatására megváltoztathatja állapotváltozóinak értékét. Az állapotváltozások leirására a névvel ellátott átmenetek (transitions) szolgálnak. Az állapotátmenet az átmenet nevéből (karakter füzér), az átmenet előfeltételéből és az állapotátmenet során végrehajtott müveletekből áll. Az átmenet előfeltétele az entitás állapotváltozóinak illetve az interfészváltozók boolean értékü függvénye (π). Amennyiben az előfeltétel igaz értékü, akkor a hozzá tartozó átmenet (α) végrehajtható. Az entitás tehát a következőképpen müködik.

Kezdetben (rendszer inicializáláskor) az entitás beállitja állapotváltozói és esetleg bizonyos interfészváltozók kezdeti értékeit. Ezután periodikusan (végtelen ciklusban) felülvizsgálja az állapotátmenetek előfeltételeit és a végrehajtható állapotátmenetek közül egyet véletlenszerüen kiválasztva valóban végrehajtja azt, aminek során az állapotváltozók és az interfészvéltozók uj értékeket kaphatnak. Az entitás tehát a kivülről jövő interakciókról az állapotátmenetek előfeltételeinek felülvizsgálatával szerez tudomást, mig saját interakcióját az interfészváltozókra történő értékadással közli környezetével. Ha az előfeltételek vizsgálatakor az entitás nem talál végrehajtható állapotátmenetet (minden átmenet előfeltétele hamis értékü), akkor a müködés ciklusa az előfeltételek ujabb felülvizsgálatával folytatódik (busy waiting).

Az állapotváltozók és interfészváltozók tipusát az absztrakt adattipus könyvtárban szereplő adattipusokra korlátozzuk. Mivel azonban az adattipus könyvtár tetszőlegesen bővithető (uj adattipus könyvtárba foglalásához csak néhány egyszerü szabály betartása szükséges), ezért ez nem igazi korlátozás. Az adattipus könyvtár jelenleg a következő tipusokat tartalmazza:

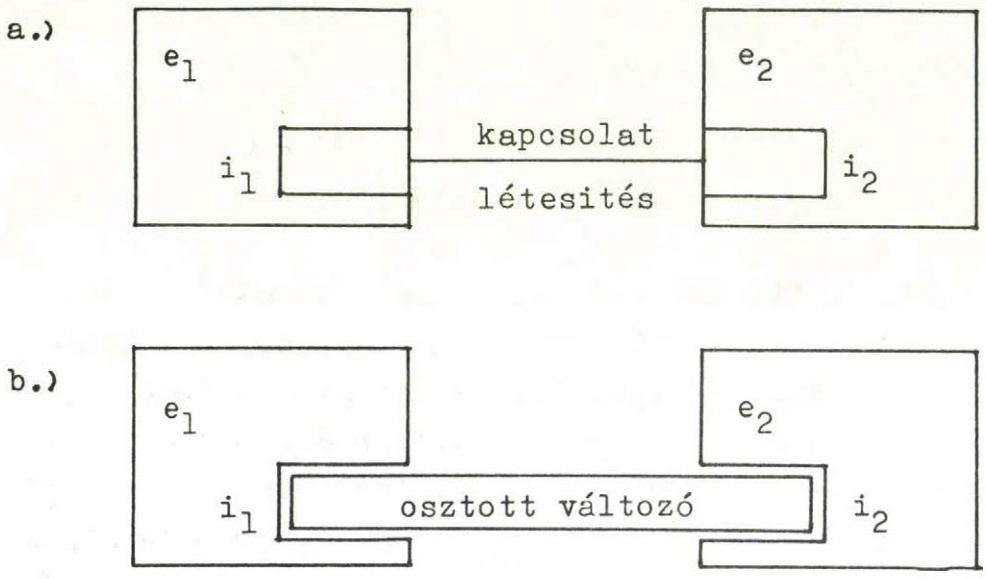
```
INTEGER, CARDINAL, CHAR, BOOLEAN,
Message, Packet, QueueOfMessage, QueueOfPacket,
HypotheticalQueueOfMessage, HistoricalQueueOfMessage,
```

Az INTEGER..BOOLEAN tipusok beépitett alaptipusok. Értékkészletük és az értelmezett müveletek megegyeznek a MODULA-2 nyelv azonos tipusainak értékkészletével és müveleteivel. A Message..QueueOfPacket tipusok a Bitalternáló protokoll leirásához használt tipusok, mig a két utolsó speciális verifikációs célu adattipus ; részletezésüket későbbre halasztjuk.

Az állapotátmenetek müveletei (actions) az állapotváltozókra és az interfészváltozókra történő értékadásokból állnak. Mint ahogy az a nyelv szintaxisából kitünik, az értékadások legkarakterisztikusabb vonása a feltételes kifejezés, IF-THEN-(ELSIF)-ELSE használata (6.5 fejezet). Egy adott müveletet alkotó összes értékadás egyidőben (szimultán) hajtódik végre, tehát a müvelet atomi jellegü. Ez azt jelenti, hogy a müvelet elvégzése előtti változó értékek alapján történik az értékadások jobb oldalának kiszámitása, majd szinkronban megtörténnek az értékadások. Mivel a jobb oldalon elhelyezkedő esetleges függvényhivások mellékhatás (side effect) mentesek, igy az értékadások tetszőleges módon felcserélhetők, sorrendjük nem befolyásolja a számitás menetét.

6.3 Az entitások közötti kapcsolatok

Az entitások közötti kommunikációt az entitások belsejéből elérhető interfészváltozók összekapcsolásával valósitjuk meg. Az interfészváltozók összekapcsolásakor az entitások közötti un. osztott változók jönnek létre. Ezt példázza a 6.6 ábra.

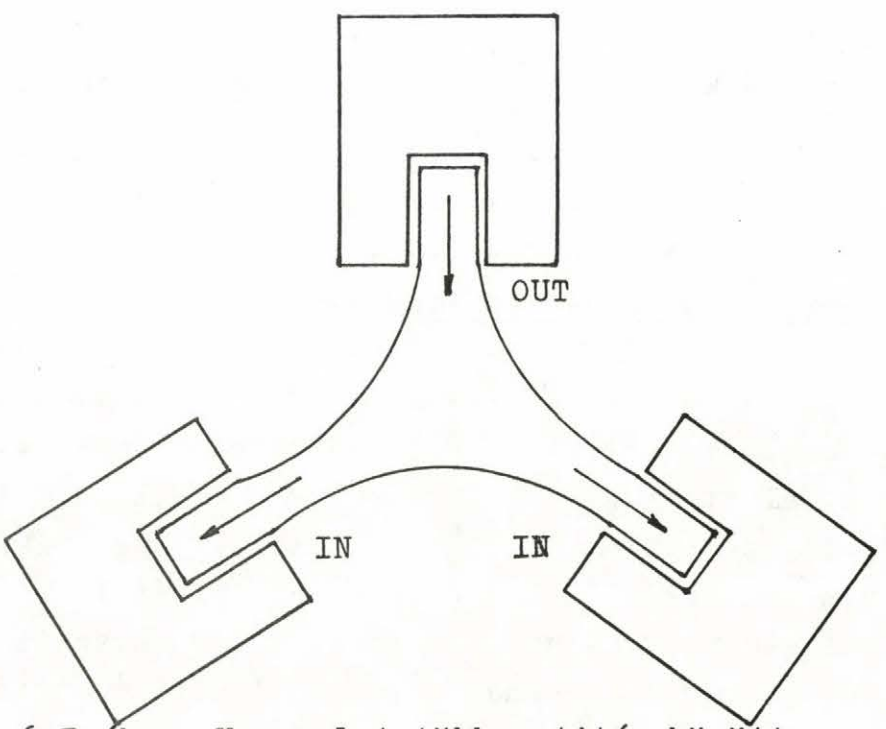


6.6 ábra Az entitások közötti kommunikáció megvalósitása

A 6.6 a.) ábra részleten az e_1 entitás i_1 és az e_2 entitás i_2 interfészváltozója között egy kapcsolat épül ki, amelynek során osztott változó jön létre (6.6 b.) ábra). Az osztott változó az e_1 entitásból i_1, az e_2 entitásból i_2 néven érhető el. Az osztott változókhoz alapvető szinkronizációs okok miatt egy időben csak egy entitás férhet hozzá. Feltételezzük tehát, hogy egy alacsonyabb szintü rejtett nyelvi mechanizmus (arbiter) kiküszöböli (feloldja) az ütközéseket, azt amikor ugyanabban az időben szeretnének az entitások az osztott változókhoz hozzáférni. Ezen alacsony szintü mechanizmustól teljes mértékben véletlenszerü müködést várunk el.

Az entitások egyenranguak, ütközés esetén azonos valószinüséggel kapják meg az osztott változókhoz való kizárólagos hozzáférés jogát. Az arbiter léte az 5. fejezetben elemezett párhuzamos rendszerek nemdeterminisztikus szekvenciális rendszerekké transzformációjának következménye. Megjegyezzük, hogy az állapotátmenet előfeltételének vizsgálata és a hozzátartozó müvelet elvégzése együttesen jelenti az osztott változóhoz való hozzáférést. Könnyü ugyanis belátni azt, hogy milyen hibák származhatnak abból, ha nem tételeznénk fel az előfeltétel vizsgálat és a müvelet oszthatatlan voltát.

Az entitások közötti kapcsolatok megadása tehát egyet jelent annak a megadásával, hogy az entitás adott interfészváltozója melyik másik entitás, melyik interfészváltozójával van kapcsolatban. Erre szolgál a specifikációs nyelv kapcsolatokat leiró része (connections). A nyelv lehetőséget ad arra, hogy ne csak két, hanem több entitás között "feszüljön ki" ugyanazon kapcsolat, tehát az osztott változóhoz több entitás is hozzáférhessen (6.7 ábra).



6.7 ábra Kapcsolat több entitás között

A kapcsolatok statikusak, rendszer inicializáláskor kiépülnek és a teljes müködés alatt fennmaradnak. A specifikációs nyelv lehetőséget teremt a kapcsolatok felépitése előtti szélesköru konzisztencia ellenőrzés végrehajtására. Csak azonos tipusu interfészváltozók között jöhet létre kapcsolat. Az egyes interfészváltozókhoz ezenkivül adatáramlási irány (IN, OUT, INOUT) is hozzá van rendelve, amely jelzi, hogy az entitás belsejéből kifelé (OUT) vagy befelé (IN) vagy mindkét irányba engedélyezett az adatáramlás. Az interfészváltozókat csak az adatáramlási irány betartásával szabad összekapcsolni. Két entitás esetén ez azt jelenti, hogy az IN változót csak OUT-tal, az OUT változót csak IN-nel illetve az INOUT változót csak INOUT tipusu változóval szabad összekapcsolni. Több entitás közötti kapcsolatok kialakitsának szabályai bonyolultabbak és megköveteli az interfészváltozókon értelmezett müveletek [IN | OUT] tipusokba sorolását. Ezzel a kérdéssel a továbbiakban terjedelmi okokból nem foglalkozhatunk. Fontos ellenőrzési lehetőséget kinál az interfészváltozókra vonatkozó kezdeti értékadás konzisztenciájának biztositása is. Az összekapcsolt interfészváltozók csak ugyanazon kezdeti értéket kaphatják, amelyet egyszerüen ugy biztositunk, hogy az összekapcsolással létrejött osztott változónak csak egy entitás adhat kezdeti értéket.

6.4 Az algoritmikus környezet fogalma

A protokoll (réteg(ek)) algoritmikus környezete a vizsgált réteg(ek) alatt és felett lévő rétegek modellje. Mint azt már a 4. fejezetben emlitettük a protokoll specifikáció és méginkább a verifikáció elengedhetetlen feltétele a környezeti modell megléte. A specifikációs nyelv bevezeti az környezet (environment) fogalmát és ezzel lehetőséget teremt

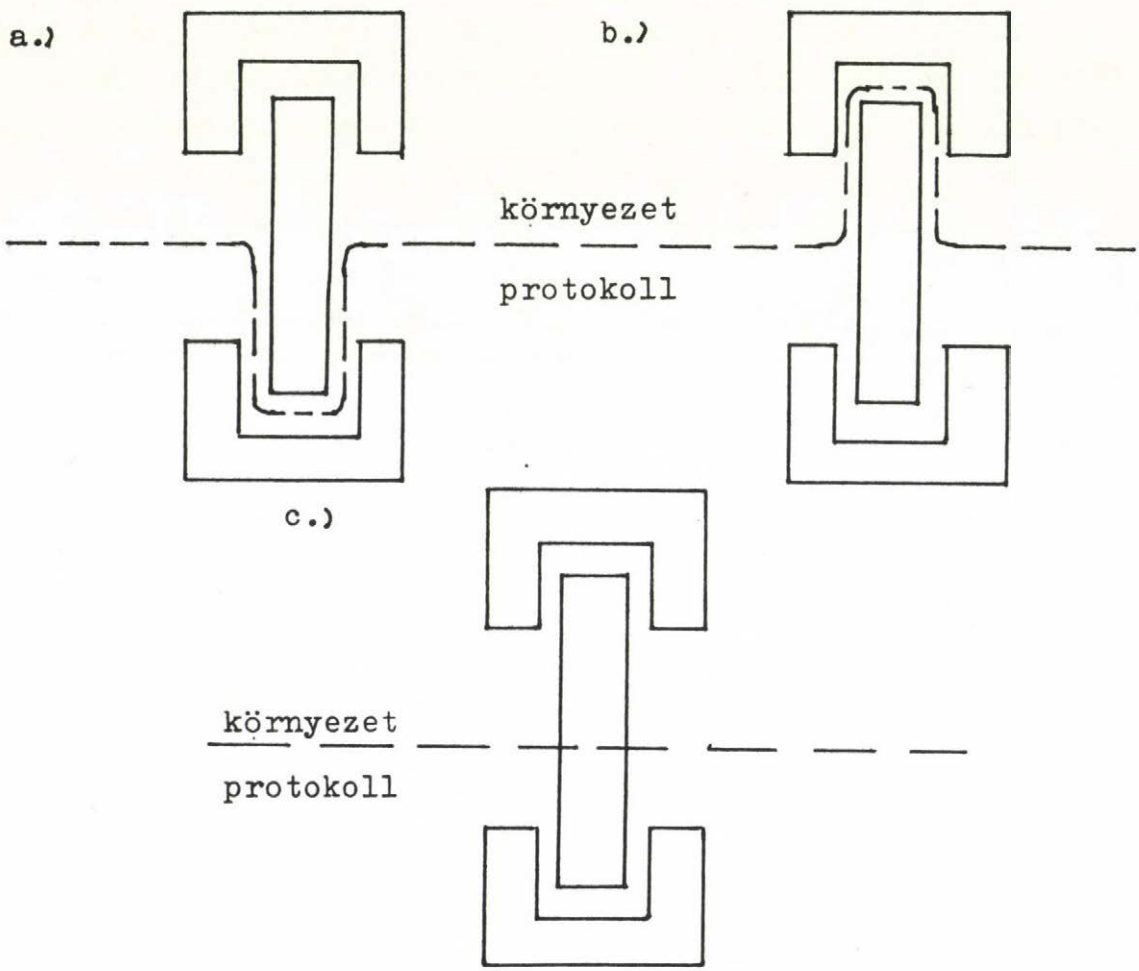
a környezeti modell formális megfogalmazására.

A környezeti modell magába süriti mindazokat a tulajdonságokat, amelyek megléte szükséges a tervezett protokoll helyes müködéséhez. A modell lehet aktiv (dinamikus), passzsiv (statikus) vagy a kettő valamilyen ötvözete. Tipikus az, amikor a tervezendő protokoll réteg alatti réteg modellje aktiv, mig a felette lévő pedig passziv. Egy ilyen példát fogunk bemutatni a 7. fejezetben. Az aktiv környezeti modell entitások segitségével irja le az alsó és felső szintek lényeges sajátságait. Két vagy több protokoll állomás közötti adatátviteli csatornák (mediumok) modellezésénél legtöbbször a medium által nem helyreállitható hibák lehetőségét fogalmazzák meg. Ilyen hiba lehet például az, hogy a medium csomagot veszthet, a szállitott csomag utközben meghibásodhat, sorrendcsere vagy egyszerüen fatális hiba léphet fel.

A passziv környezeti modell esetén a környezetet a vizsgált illetve az alatta és felette lévő rétegek közötti kommunikációra szolgáló interfészváltozóknak adott kezdeti értékek alkotják. Mindezt legtöbbször az teszi lehetővé, hogy általában a szintek közötti kapcsolatot várakozási sorok képezik (például az elszállitásra váró csomagok sora). Ezen sorokat bizonyos szintig feltöltve megteremtődik a protokoll müködésének feltétele (például megindul a csomag szállitás). A müködés természetesen egy idő után abbamarad (mert vagy kiürül a sor vagy valamilyen hiba lép fel). A passziv modell korlátozott leiró képességü, használata néha még a protokoll és a környezet közötti interfész megváltoztatását is megköveteli.

Itt kell foglalkoznunk azzal a kérdéssel, hogy rétegek közötti osztott változók (például sorok) hova tartoznak. A kérdésnek a verifikáció szempontjából van nagy jelentősége hiszen ez határozza meg, hogy az osztott változót a protokoll vagy környezet állapotánál kell-e számitásba venni. A

6.8 ábrán feltüntettük az alapvető lehetőségeket. Az a.) részleten az osztott változó teljes egészében a környezethez, a b.) ábrán a protokollhoz tartozik. A c.) részlet egy köztes megoldást mutat, melyben az (állapot szempontjából részekre bontható) osztott változó egy része a protokollhoz, másik része a környezethez tartozik. A 7. fejezet példájában ilyen megoldást fogunk alkalmazni.



6.8 ábra A protokoll és a környezet közötti interfészváltozók lehetséges helyzetei

Már korábban emlitettük, hogy a nyelv opcionálisan lehetővé teszi a protokoll és környezete közötti határvonal éles meghatározását is. Az interfészváltozók az exportált

(EXPORTED) csoportba tartoznak, ha keresztül haladnak a környezetet és a specifikált protokollt elválasztó határfelületen. Ellenkező esetben az interfészváltozók belső (INTERNAL) tipusuak.

A környezet leirás különleges eleme a protokoll várakozás (WAIT) explicit meghatározására szolgáló rész. Ebben a részben a protokoll tervezője a környezeti (EXPORTED) interfészváltozókra felirt predikátummal meghatározhatja azokat a helyzeteket, amelyben a protokoll megállása nem deadlocknak, hanem a felső szint interakciójára való várakozásnak minősül. Felhasználása csak passziv környezeti modell esetén indokolt.

A protokoll tulajdonságok (7.1 fejezet) formális megfogalmazását szolgálja a specifikációs nyelv tulajdonságokat (properties) leiró része, melyben névvel ellátott predikátum függvények definiálása lehetséges. A predikátumok protokoll invariánsok kifejezési formái. A 9. fejezet verifikációs rendszere ezen invariánsok érvényességi analizisével ellenőrizni tudja a specifikált tulajdonságok meglétét. Az invariánsok megfogalmazását megkönnyiti az absztrakt adattipus könyvtárban szereplő HistoricalQueueOfMessage segéd (história) változó bevezetése. A história változókat az entitások állapotváltozói között deklaráljuk; ugyanazon szabályok vonatkoznak rájuk is mint az állapotváltozókra. A protokoll müködésében ténylegesen nem vesznek részt (például nem hordoznak állapot információt sem). Feladatuk kizárólag a protokolltól elvárt tulajdonságok megfogalmazásának egyszerüsitése. A história változók nevüket onnan kapták, hogy magukba gyüjtik a protokoll müködéséről szóló globális információkat, a müködés történetét.

6.5 A specifikációs nyelv szintaxisa

A specifikációs nyelv szintaxis leirása az EBNF (Extended Backus-Naur Form) formalizmust követi. A nemterminálisok értelmes magyar szavak, mig a terminálisok vagy csupa nagybetüből álló angol szavak vagy idézőjelbe tett karakter füzérek. A szintaxis szabályok

$S = E .$

formájuak, ahol S valamely szintaktikus egység, E pedig

$T_1 | T_2 | \ldots | T_n \qquad (n>0)$

alaku kifejezés. A T_i tagok

$F_1 F_2 \ldots F_m \qquad (m>0)$

tényezőkből állnak. A T_i tehát az F_i tényezők konkatenációja. Az F_i tényezők terminálisok vagy nemterminálisok esetleg $[E]$ vagy $\{E\}$ alakuak. A szögletes zárójel a nulla vagy egyszeri, a kapcsos zárójel a nulla, egy, kettő, ... stb. többszörözést jelenti.

```
Azonosito = Betu { Betu | Szamjegy } .

Szam  = Integer | Real .

Integer = Szamjegy { Szamjegy } .

Real = Szamjegy { Szamjegy } "." { Szamjegy } [ SkalaTenyezo ] .

SkalaTenyezo = "E" [ "+" | "-" ] Szamjegy { Szamjegy } .

Szamjegy = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .

Relacio = "=" | "≠" | "<>" | "<" | "<=" | ">" | ">=" .

AdditivOperator = "+" | "-" | OR .

MultiplikativOperator = "*" | "/" | DIV | MOD | AND .

AzonositoLista = Azonosito { "," Azonosito } .

RendszerDeklaracio = SYSTEM RendszerNev ";" RendszerKomponensek
      END RendszerNev "." .

RendszerNev = Azonosito .

RendszerKomponensek = { EntitasLeiras } KornyezetLeiras
      KapcsolatLeiras [ KovetelmenyLeiras ] .

EntitasLeiras = ENTITY EntitasNev ";" EntitasKomponensek
      END EntitasNev ";" .

EntitasNev = Azonosito .

EntitasKomponensek = [ AllapotValtozoDeklaracio ] InterfeszValtozoDeklaracio
      AtmenetDeklaracio MuveletDeklaracio [ KezdetiAllapotDeklaracio ] .

AllapotValtozoDeklaracio = STATE VARIABLES { ValtozoDeklaracio ";" } .

ValtozoDeklaracio = AzonositoLista ":" Tipus .

Tipus = Azonosito .

InterfeszValtozoDeklaracio = INTERFACES { [ EXPORTED | INTERNAL ]
      { AdatAramlasIrany ValtozoDeklaracio ";" } } .

AdatAramlasIrany = [ IN | OUT | INOUT ] .

AtmenetDeklaracio = TRANSITIONS { AtmenetNev ":" PRECOND Kifejezes ";" } .
```

```
AtmenetNev = Azonosito .

Kifejezes = EgyszeruKifejezes [ Relacio EgyszeruKifejezes ] .

EgyszeruKifejezes = [ "+" | "-" ] Tag { AdditivOperator Tag } .

Tag = Tenyezo { MultiplikativOperator Tenyezo } .

Tenyezo = Szam | [ EntitasNev "." ] Azonosito [ AktualisParameterek ] |
     "(" Kifejezes ")" | NOT Tag | "'" Karakter "'" |
     IF Kifejezes THEN Kifejezes { ELSIF Kifejezes THEN Kifejezes }
     ELSE Kifejezes END .

AktualisParameterek = "(" [ KifejezesLista ] ")" .

KifejezesLista = Kifejezes { "," Kifejezes } .

MuveletDeklaracio = ACTIONS { AtmenetNev ":" { Ertekadas ";" }
     END AtmenetNev ";" } .

Ertekadas = Azonosito ":=" Kifejezes .

KezdetiAllapotDeklaracio = INITIAL STATES { Ertekadas ";" } .

KornyezetLeiras = ENVIRONMENT { EntitasLeiras } [ InterfeszValtozoDeklaracio ]
     [ KezdetiAllapotDeklaracio ] [ WAIT Kifejezes ";" ] .

KapcsolatLeiras = CONNECTIONS { Elem { "," Elem } "<=>"  Elem { "," Elem } } .

Elem = [ EntitasNev "." ] InterfeszValtozoNev .

InterfeszValtozoNev = Azonosito .

KovetelmenyLeiras = PROPERTIES { TulajdonsagNev ":" Kifejezes ";" } .

TulajdonsagNev = Azonosito .
```

7. EGY EGYSZERÜ PÉLDA A JAVASOLT MÓDSZER ALKALMAZHATÓSÁGÁNAK BEMUTATÁSÁRA

A 6. fejezetben ismertetett protokoll specifikációs nyelv alkalmazásaként bemutatjuk a Bitalternáló protokoll egyszerüsitett változata specifikációját. Először a protokoll müködését szövegesen irjuk le, majd megadjuk a formális leirást is.

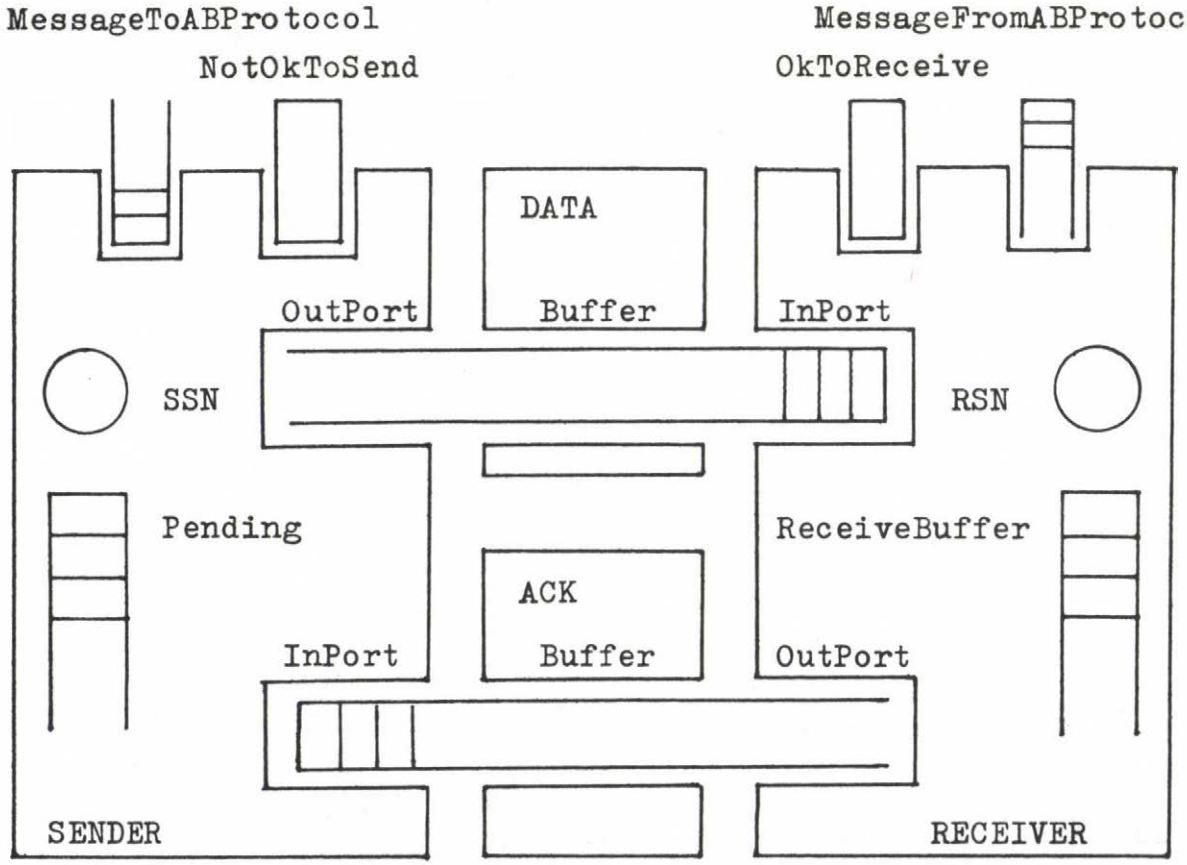
7.1 A Bitalternáló protokoll

A Bitalternáló protokoll a sorrendhelyességet nem biztositó protokollok családjába tartozik [Davies és m. 82]. Az ARPA illetve az EIN hálózatokban használt adatkapcsolat szintü protokoll üzeneteket szállit megbizhatatlan adatátviteli csatornán keresztül. Logikai csatornákat definiál és ezeken egymástól függetlenül egy-egy csomagot továbbit. A csomag egy bites sorszámból, a logikai csatorna számából és az üzenetből áll. A protokoll például Ø sorszámmal elküldött csomag után akkor küldheti a következő csomagot (1 sorszámmal), ha timeout időn belül hibátlanul megérkezik az előző csomag nyugtája. Ha a nyugta megérkezése előtt időhatár tullépés (timeout) következik be, akkor a protokoll ujra küldi az elküldött üzenet másolatát. A mechanizmus a pozitiv nyugtázás, automatikus ismétlés néven ismert. A vételi oldalon a helyesen (helyes sorszámmal) érkezett csomag üzenetrészét a felsőbb szint felé továbbitják (ha az fogadni tudja) és nyugtázásképpen a sorszámot visszaküldik. A nyugta csomag üzenetrésze ellenkező irányban szállitott üzenetet hordozhat, mi azonban a későbbiekben - mivel csak egyetlen logikai csatornát specifikálunk - nyugtázásképpen a megérkezett csomag változatlan visszaküldését alkalmazzuk. Duplex megvalósitás esetén mindkét irányban számtalan logikai csatornát definiálnak, ami ugyan növeli a hatékonyságot, de

sorrendhelyességet nem biztosit. A protokoll részletesebb leirása helyett az irodalomra utalunk.

7.2 A protokoll specifikációja

A protokoll leirásához két entitást, a küldő (Sender) és a fogadó (Receiver) entitást használtunk. A 7.1 ábrán feltüntettük a protokoll specifikációjának illetve környezeti modelljének vázlatát. Az ábrán a Sender és Receiver protokoll entitásokon kivül az alsó rétegeket modellező Data és Ack medium entitások is megtalálhatók. A környezeti modell entitásai a megbizhatatlan átviteli csatorna csomagvesztő tulajdonságát modellezik (csomag duplikálódást illetve egyéb meghibásodást ez esetben nem tételezünk fel).



7.1 ábra A Bitalternáló protokoll és környezete

A Sender entitás a MessageToSend és az MessageInTransit interfészváltozókon keresztül kapcsolódik a felső szinthez. A környezet felől ezeket a változókat a MessageToABProtocol illetve a NotOkToSend néven érhetjük el. A MessageToSend üzenetsor a felső szinttől kapott üzeneteket hordozza. A MessageInTransit BOOLEAN tipusu változó jelzi, ha valamely üzenet éppen szállitás alatt van. A Sender entitás két állapotváltozót foglal magában. Az SSN BOOLEAN tipusu változó az aktuális egy bites sorszámot tartalmazza, mig a Pending csomagsor az elküldött de még nem nyugtázott csomagok puffere. Itt jegyezzük meg, hogy a csomagformátumban nem szerepel a logikai csatorna sorszáma, mivel csak egyetlen csatorna müködését specifikáljuk. A Sender tartalmaz még egy Sent nevü história változót (üzenetsort) is, amely a felső szinttől szállitásra kapott üzenetek mindegyikét eltárolja. Az alsó szinttel való kapcsolat tartást teszik lehetővé az OutPort és InPort interfészváltozók. A Sender UserSend átmenete akkor válik végrehajthatóvá, ha van szállitásra váró üzenet a MessageToSend sorban (a sor nem üres) és befejeződött a korábbi üzenet továbbitása (a MessageInTransit hamis értékü). Ekkor az üzenetet kiegészitve az aktuális SSN sorszámmal az OutPorton keresztül az entitás csomagként elküldi és felkészül az esetleges ujraküldésre is (a Pending pufferbe eltárolja az elküldött csomagot). Az időkezelésre a specifikációban a TimeOut átmenet szolgál. Időhatár tullépés csak akkor következhet be, ha a puffer nem üres, tehát egy csomagot elküldtek, de még nem érkezett meg a nyugtája. Timeout esetén a puffer tartalma az OutPortra kerül. A Sender Receive átmenete a nyugta üzenetek fogadását végzi. Ha a nyugta csomag a várt sorszámot hordozza, akkor az entitás belső puffere felszabadul és lehetővé válik az ujabb üzenet elküldése (természetesen uj sorszámmal).

A Receiver entitás belső felépitése hasonlit a Sender entitásához. Az RSN BOOLEAN tipusu állapotváltozó az aktuális sorszámot, a ReceiveBuffer a megérkezett de a felsőbb

szint felé még nem továbbitott csomagokat hordozza. A Received história változó (üzenetsor) a protokoll tulajdonságok megfogalmazását megkönnyitő segédváltozó. A felső szint a UserWishesToReceive (OkToReceive) BOOLEAN változón keresztül közli az entitással, hogy képes fogadni egy esetlegesen megérkezett üzenetet. Ekkor az a MessageReceived (MessageFromABProtocol) interfészváltozóba kerül a UserRcv átmenet segitségével. A Receiver entitás Receive átmenete az InPorton keresztül érkezett csomagokat fogadja. Ha a csomag a helyes (várt) sorszámmal érkezik, akkor az a ReceiveBuffer pufferbe kerül. A Receive átmenet különleges ujra szinkronizáló mechanizmust is tartalmaz, amelyre később még kitérünk.

A Data és Ack medium entitások az adatátviteli csatornák csomagvesztő tulajdonságát modellezik a LosePacket átmenetük segitségével. A mediumok az első pillanatban szokatlan módon nem tartalmaznak állapotváltozót. Az egyetlen interfészváltozójuk össze van kapcsova a Sender és Receiver entitások megfelelő (InPort, OutPort) interfészváltozóival, igy létrejönnek a három (Sender - Data - Receiver illetve Receiver - Data - Sender) entitás által közösen használt un. osztott változók. Ezek az osztott változók hordozzák az entitások által kibocsájtott csomagokat. Mivel a rendszerünk nemdeterminisztikus szekvenciális (5. fejezet), igy a müködés lényege az, hogy egy entitás kibocsájtotta csomag vagy elvész a medium entitások állapotváltozása révén vagy pedig hibátlanul megérkezik rendeltetési helyére. A medium entitások bonyolultabb megfogalmazásával természetesen lehetőség van egyéb, itt nem feltételezett tulajdonság modellezésére is (például korlátozott csatorna kapacitás stb.), amelytől most az egyszerüség kedvéért eltekintettünk.

A környezeti modell a Data és Ack entitásokon kivül még a felső protokoll szint két interfészváltozójának deklarációját is tartalmazza a megfelelő kezdeti értékekkel (verifikációs célból).

A tulajdonságokat leiró részben az invariáns azt fejezi ki, hogy a Sender entitás által elküldött csomagok mindegyike megérkezik sorrendhelyesen a Receiver entitáshoz. Ez az invariáns tulajdonság a logikai csatorna protokoll helyességét formalizálja.

A protokollba beépitett ujra szinkronizáló mechanizmus lényege az, hogy ha valamilyen oknál fogva az SSN és RSN értékei kezdetben nem egyeznek meg (hibás inicializálás), akkor ezt a protokoll egy üzenet elvesztése árán feloldja, lásd. Receiver entitás Receive átmenete.

```
(*************************************************************
*                                                           *
*          Alternating Bit Protocol Specification           *
*                                                           *
*                                                           *
*                                                           *
*************************************************************)

SYSTEM AlternatingBitProtocol;


ENTITY Sender;

STATE VARIABLES
  SSN     :BOOLEAN;
  Pending:QueueOfPacket;
  Sent    :HystoricalQueueOfMessage;

INTERFACES EXPORTED
    MessageInTransit:BOOLEAN;
    MessageToSend    :QueueOfMessage;
    InPort,OutPort   :QueueOfPacket;


TRANSITIONS
  UserSend: PRECOND NOT Empty(MessageToSend) AND
    NOT MessageInTransit;

  TimeOut : PRECOND NOT Empty(Pending);
  Receive : PRECOND NOT Empty(InPort);
```

```
ACTIONS
  UserSend:
    Sent:=Add(Sent,Front(MessageToSend));
    Pending:=Add(Pending,
      MakePacket(Front(MessageToSend),SSN));
    MessageInTransit:=TRUE;
    OutPort:=Add(OutPort,
      MakePacket(Front(MessageToSend),SSN));
    MessageToSend:=Remove(MessageToSend);
  END UserSend;

  TimeOut:
    OutPort:=Append(OutPort,Pending);
  END TimeOut;

  Receive:
    SSN:=IF SeqNumber(Front(InPort))=SSN
      THEN NOT SSN ELSE SSN END;

    MessageInTransit:=IF SeqNumber(Front(InPort))=SSN
      THEN FALSE ELSE MessageInTransit END;

    Pending:=IF SeqNumber(Front(InPort))=SSN
      THEN Clear(Pending) ELSE Pending END;

    InPort:=Remove(InPort);
  END Receive;

INITIAL STATES
  SSN:=FALSE;
  Pending:=NewQueueOfPacket();
  Sent:=NewHystoricalQueueOfMessage();
  MessageInTransit:=FALSE;
  OutPort:=NewQueueOfPacket();
END Sender;


ENTITY Receiver;

STATE VARIABLES
  RSN           : BOOLEAN;
  ReceiveBuffer: QueueOfPacket;
  Received      : HystoricalQueueOfMessage;

INTERFACES EXPORTED
    UserWishesToReceive:BOOLEAN;
    MessageReceived     :QueueOfMessage;
    InPort,OutPort      :QueueOfPacket;


TRANSITIONS
  UserRcv:PRECOND NOT Empty(MessageReceived) AND
    UserWishesToReceive;

  Receive:PRECOND NOT Empty(InPort);
```

```
ACTIONS
  UserRcv:
    Received:=Add(Received,Front(ReceiveBuffer));
    MessageReceived:=Remove(MessageReceived);
    UserWishesToReceive:=FALSE;
    OutPort:=Append(OutPort,ReceiveBuffer);
    ReceiveBuffer:=Remove(ReceiveBuffer);
  END UserRcv;

  Receive:
    ReceiveBuffer:=IF SeqNumber(Front(InPort))=RSN THEN
      Add(ReceiveBuffer,Front(InPort)) ELSE ReceiveBuffer END;

    MessageReceived:=IF SeqNumber(Front(InPort))=RSN THEN
      Add(MessageReceived,Front(InPort)) ELSE MessageReceived END;

    OutPort:=IF (SeqNumber(Front(InPort))()RSN) AND
      Empty(ReceiveBuffer) THEN
      Add(OutPort,Front(InPort)) ELSE OutPort END;

    RSN:=IF SeqNumber(Front(InPort))=RSN THEN
      RSN:=NOT RSN ELSE RSN END;

    InPort:=Remove(InPort);
  END Receive;

INITIAL STATES
  RSN           :=FALSE;
  ReceiveBuffer :=NewQueueOfPacket();
  Received      :=NewHystoricalQueueOfMessage();
  MessageReceived:=NewQueueOfMessage();
  OutPort       :=NewQueueOfPacket();

END Receiver;


ENVIRONMENT

ENTITY Data;

INTERFACES EXPORTED
  Buffer:QueueOfPacket;

TRANSITIONS
  LosePacket:PRECOND NOT Empty(Buffer);

ACTIONS
  LosePacket:
    Buffer:=Remove(Buffer);
  END LosePacket;

END Data;
```

```
ENTITY Ack;

INTERFACES EXPORTED
  Buffer:QueueOfPacket;

TRANSITIONS
  LosePacket:PRECOND NOT Empty(Buffer);

ACTIONS
  LosePacket:
    Buffer:=Remove(Buffer);
  END LosePacket;

END Ack;

INTERFACES EXPORTED
  MessageToABProtocol,MessageFromABProtocol:QueueOfMessage;
  NotOkToSend,OkToReceive:BOOLEAN

INITIAL STATES
  MessageToABProtocol:=Add(Add(Add(MessageToABProtocol,
    MakeMessage(1)),MakeMessage(1)),MakeMessage(1));
  MessageFromABProtocol:=NewQueueOfMessage();
  OkToReceive:=TRUE;

WAIT
  Empty(MessageToABProtocol) OR (NOT OkToReceive);

CONNECTIONS
  Sender.OutPort <=> Data.Buffer, Receiver.InPort;
  Receiver.OutPort <=> Ack.Buffer, Sender.InPort;
  MessageToABProtocol <=> Sender.MessageToSend;
  NotOkToSend <=> Sender.MessageInTransit;
  MessageFromABProtocol <=> Receiver.MessageReceived;
  OkToReceive <=> Receiver.UserWishesToReceive;
```

```
PROPERTIES
  Correctness:
      EqualQueue(Sender.Sent,Receiver.Received,
        IF (Sender.SSN()Receiver.RSN) AND
          Empty(Receiver.MessageReceived) OR
          (Sender.SSN=Receiver.RSN) AND
          Empty(Sender.Pending)
        THEN
          Length(Sender.Sent)
        ELSE
          Length(Sender.Sent)-1
        END)
    AND
      EqualMessage(Back(Sender.Sent),
        IF (Sender.SSN()Receiver.RSN) AND
          Empty(Receiver.MessageReceived) OR
          (Sender.SSN=Receiver.RSN) AND
          Empty(Sender.Pending)
        THEN
          Back(Sender.Sent)
        ELSE
          Text(Front(Sender.Pending))
        END);

END AlternatingBitProtocol.
```

8. PROTOKOLLOK TULAJDONSÁGAINAK ELLENŐRZÉSE ÁLLAPOT-ELÉRHETŐSÉGI VIZSGÁLATTAL

A fejezetben a protokoll verifikáció egy "hibrid" módszerét, a bővitett elérhetőségi analizist vezetjük be. A formális protokoll verifikáció módszere nagymértékben függ a protokoll specifikáció során alkalmazott formalizmustól. A 6. fejezetben specifikációs nyelvet javasoltunk protokoll leirásra. A specifikációs nyelv megpróbálta egyesiteni magában az állapotátmeneten és a programozási nyelveken alapuló eljárások előnyös vonásait. Ez alapján természetes módon vetődik fel az a gondolat, hogy a specifikációs nyelvünkhöz illeszkedő verifikációs módszer is egyesitse a két különböző verifikációs irányzat (állapotelérhetőségi vizsgálat és programhelyesség bizonyitás) előnyös vonásait. A megoldást az elérhetőségi analizis protokoll invariánsok érvényességi vizsgálatával történő kiegészitésében találtuk meg. Az ilyen módon bővitett elérhetőségi analizis ismertetése előtt a protokoll tulajdonságokat, azok értelmezését tárgyaljuk először szöveges formában, majd felhasználva az 5. fejezet matematikai modelljét megadjuk a főbb tulajdonságok formális megfogalmazását is. A bővitett elérhetőségi vizsgálat ezen tulajdonságok felderitését teszi lehetővé. A módszer számitógéppel támogatott megvalósitását a 9. fejezetben fogjuk részletesen bemutatni.

8.1 Protokoll tulajdonságok

A protokollok verifikálható tulajdonságait két fő csoportba, az általános és a specifikus tulajdonságok csoportjába sorolhatjuk.

A specifikus tulajdonságok közé a különféle protokollokat egyedileg jellemző, a többi protokolltól megkülönböztető tulajdonságok tartoznak. Ezek legfontosabbika a funkcionális helyesség tulajdonság, amely azt fejezi ki, hogy a protokoll kielégiti a szolgáltatás specifikációját. Lényegében tehát a szolgáltatás specifikáció által előirt szolgáltatások alkotják a specifikus tulajdonságok halmazát. Például transzport protokoll esetén a transzport connection létesités a protokoll specifikus tulajdonsága. Mint már korábban emlitettük a funkcionális helyesség formális megfogalmazása a protokollokra egyedi rendszer invariánsok felállitásával történhet. Ezen predikátumok protokoll müködéssel szembeni invarianciájának bebizonyitása (ellenőrzése) a funkcionális helyesség belátását jelenti.

Az általános tulajdonságok a protokollok mindegyikét (vagy inkább nagyobb halmazát) jellemző tulajdonságok. Lényegében a szolgáltatás specifikációk implicit részének tekinthetők. Az alábbiakban felsoroljuk (a teljesség igénye nélkül) a nézetünk szerint fontosabb általános tulajdonságokat. Nem állitjuk, hogy ezek mindegyike független egymástól, inkább "fenomenologikus" protokoll tulajdonságoknak tekinthetők, amelyek "mikro szintü" kapcsolata nem ismert. Azt sem állitjuk, hogy az itt következő fogalmak egyetlen helyes értelmezését adjuk meg. Az irodalomban gyakran ellentmondó "definiciókat" (szöveges értelmezéseket) találhatunk, aminek fő oka a tulajdonságok valamely elméleti kereten (formalizmuson) belüli formális értelmezésének hiánya. A terminológiai zürzavar tipikus példája az, amikor ugyanazon tulajdonságot különböző neveken, vagy különbözőket azonos módon neveznek. Nagy problémát jelent az is, hogy a fogalmak jelentős része a programozás (különösen a paralell programozás) elmélet területéről származik. A protokollok párhuzamos programokkal történő implementációja összemossa a valódi protokoll tulajdonságokat az implementáció, a paralell programok bizonyos sajátságaival. Véleményünk szerint a terminológiai kavarodás, eltérő értelmezések feloldásának egyet-

len utja a fogalmak szigoru formális definiciója egy eléggé általános elméleti kereten belül. Megtalálva a különféle protokoll modellek közötti összefüggést (például leképezés formájában) a protokoll tulajdonságok definiciói is áttranszformálhatókká válnak, megteremtve ezzel a dolgozat mottójának választott konszenzust. A legfontosabb általános tulajdonságok a következők:

- teljesség
- deadlock mentesség
- liveness
- progress (livelock, tempo-blocking mentesség)
- helyreállithatóság
- stabilitás
- önszinkronizáció
- terminálódás (ciklikusság)
- korlátozások.

A fogalmak vázlatos értelmezéséhez a 6. fejezet specifikációs nyelvének elemeit fogjuk felhasználni, feltételezve, hogy a mögöttes mechanizmus az 5. fejezet állapotátmenet rendszere.

A protokoll specifikáció teljessége (completeness) azt fejezi ki, hogy a rendszer entitás minden a környezetéből származó interakciót észlel és szükség esetén válaszol rá (például állapotátmenettel). A specifikáció teljességi hiányait különféle csoportokba sorolhatjuk. Az első, legegyszerübb csoportot a protokoll leirás szintaktikus és statikus szemantikai hibái alkotják. Abban az esetben ha például egy entitás egyik deklarált interfészváltozója egyetlen egy állapotátmenetének előfeltételében sem szerepel, akkor az entitás nem fogja észlelni a kérdéses interfészváltozó "szállitotta" interakciókat. A jelenség oka vagy az interfészváltozó felesleges deklarációja (például nem áll kapcsolatban egyetlen más interfészváltozóval sem), tulspecifikáltság vagy hiányos specifikáció (vagy a megfelelő átmenet hiányzik, vagy csak valamelyik előfeltétel megfogalmazása hibás). Ezen és a hasonló hibákat (például tipus inkompati-

bilitás) a specifikációs nyelv elemzője még fordítási időben ki tudja mutatni. Sokkal nehezebb azonban azon specifikációs hiányokat kimutatni, amelyek nem járnak együtt a leirás hibájával. Ilyenkor az elemző hibátlan specifikációt jelez, a specifikáció mégis hiányos. A protokoll tervező például nem készitette fel a protokollt az alacsonyabb szintek bizonyos müködési zavarainak (pl. csomag meghibásodás, csomag vesztés stb.) lekezelésére, vagy például nem vette figyelembe az ütközések lehetőségét (pl. hivás ütközés). Az ilyen tipusu hibák felderitése csak valamilyen verifikációs módszer alkalmazásával képzelhető el.

A protokoll rendszerek legalapvetőbb elvárt tulajdonsága a deadlock mentes müködés. A rendszer akkor kerül holtpont állapotba, ha valamely belső ok miatt a rendszer egyetlen egy entitásának sincs végrehajtható állapotátmenete. Ekkor kialakul egy entitásokból álló zárt hurok, amelyben minden entitás a megelőző entitás interakciójára vár. A belső ok feltétel azért szükséges a megfogalmazásban, hogy kizárjuk a felső szintek okozta protokoll várakozásokat a deadlock szituációk köréből. A deadlock helyzetek leggyakrabban a protokoll nem teljes specifikációja miatt következnek be. Elképzelhető, hogy nem a teljes protokoll rendszer, hanem annak csak bizonyos része (bizonyos entitás csoport) blokkolódik örökre. Ekkor a rendszer csökkentett funkciókkal esetleg még tovább müködhet, megnehezitve ezzel a részleges holtpont állapot korai felismerését. Élesen megkülönböztetjük a protokoll implementációban előforduló deadlock helyzeteket a valódi protokoll deadlockoktól. Az implementációban előálló holtpont helyzetek ugyanis nemcsak valódi protokoll deadlock miatt, hanem például helytelen erőforrás allokáció stb. miatt is bekövetkezhetnek. Nem tekintjük deadlock helyzetnek az explicit végállapottal terminálódó protokollok végállapotát.

A liveness tulajdonság azt fejezi ki, hogy bármely elérhető állapotból bármely másik állapot elérhető (8.2 feje-

zet). A live tulajdonságot nemcsak rendszerre, hanem egyedi állapotra (vagy átmenetre) is értelmezni szokták. Egy adott állapot (vagy átmenet) akkor live tulajdonságu, ha a rendszer kezdeti állapotából elérhető állapotokból elérhető [Bochmann 78]. Általában a protokollok olyan felépitésüek, hogy van bennük egy kitüntetett un. "home" állapot, amely live tulajdonságu és amelyből minden protokoll funkció végrehajtható.

A progress (livelock, tempo-blocking freeness) sajátság a protokoll felesleges (!) végtelen ciklusoktól való mentességét fejezi ki. Elképzelhető ugyanis az, hogy a rendszer entitásainak egy csoportja interakciók olyan végtelen ciklusába kerül, amely meggátolja a helyes müködést. A vonalszakadáskori állandó ismétlés tipikus egyszerü példa erre. Az, hogy a protokoll ciklus mikor felesleges, vagyis mikor nem végez a protokoll "hasznos munkát", nehéz formalizálni és ebből következőleg a ciklusok automatikus osztályozása szinte megoldhatatlan feladatnak látszik. (Ezért például a 9. fejezet számitógéppel támogatott verifikáló rendszerében is az ember feladata a ciklusok kivánatos, vagy nem kivánatos voltának eldöntése.) Az állapotátmeneten alapuló protokoll leirási módszerek esetén általánosságban igaz az a megállapitás, hogy nehézkes a progress tulajdonság formalizálása. Talán ez volt az egyik fő oka a temporális logikák specifikációs célokra való kisérleti felhasználásának is, ahol tudvalevőleg "természetes módon" lehet megfogalmazni a progress sajátságot.

A helyreállithatóság (recoverability) fogalma arra a szemléletre épül, amely a protokoll állapotterét két részre, a normális és a rendellenes müködési állapottérre osztja. A rendellenes állapottérbe azon protokoll állapotok tartoznak, amelyekbe valamely hiba folytán kerül a rendszer. A rendszer akkor helyreállitható, ha véges számu lépésben, véges idő alatt az abnormális müködési állapottér bármely állapotából képes visszatérni a normális müködés zónájába. A megfogal-

mazás alapproblémája a hiba fogalmának intuitiv definiciója. Sok esetben ugyanis önkényes az, hogy mit tekintünk normális és mit abnormális protokoll müködésnek.

A protokoll stabilis tulajdonságu, ha valamely hiba kiváltó okának megszünése után a helyreállitáshoz szükséges lépések (állapotátmenetek) száma nem növekszik a müködés során. A lépesszám növekedése ugyanis egy idő után gyakorlatilag megakadályozza a hibákból való felélledést, elfogadhatatlanul lassuvá téve azt, bár a helyreállithatóság még mindig fennállhat, hiszen nagy de azért mindig véges számu állapotátmenet zajlik le a normális müködési tartományba való visszatéréskor. A helyreállithatóság tehát a stabilitás szükséges de nem elégséges feltétele.

Az önszinkronizációs (self synchronization) protokoll tulajdonság a helyreállithatóság egy fajtája. Azokat a protokollokat szokták önszinkronizálónak mondani, amelyek nem megfelelő inicializálás esetén is véges sok lépésben, véges idő alatt eljutnak a helyes müködés tartományába.

A protokollok vagy ciklikus müködésüek, vagy explicit végállapotban terminálódnak.

A korlátozások (boundness) alatt a protokollba beépitett tevékenység számra előirt korlátokat értjük (pl. maximális ismétlési szám). A puffer méretek, csatorna kapacitások ugyancsak fontos figyelembe veendő korlátok.

8.2 A protokoll tulajdonságok formális megfogalmazása

A protokollok (Q,R,N,P) matematikai modellje (5. fejezet) alapján bevezetjük az elérhetőségi reláció fogalmát. Az elérhetőségi reláció az R reláció reflexiv és tranzitiv lezárása.

Definició 8.1 : A (Q,R,N,P) átmenet rendszer Q állapot halmazán értelmezett elérhetőségi reláció (R^*) olyan bináris reláció (jelölés: $\xrightarrow{*}$) melyre:

1. $\forall q, q' \in Q$-ra ha $q \rightarrow q' \in R$ akkor $q \xrightarrow{*} q'$;
2. $\forall q \in Q$-ra $q \xrightarrow{*} q$;
3. $\forall q, q', q'' \in Q$-ra ha $q \xrightarrow{*} q'$ és $q' \xrightarrow{*} q''$ akkor $q \xrightarrow{*} q''$;
4. R^* minden eleme az 1. - 3. szabályok véges számu alkalmazásával előállitható. □

Egy állapot tehát akkor elérhető egy másik állapotból, ha van a másik állapotból kiinduló olyan rendszer müködés (5.7 definició), amelyik tartalmazza a kérdéses állapotot. A későbbiekben bennünket egy kitüntetett állapotból, a $q_0 \in Q$ kezdeti állapotból elérhető állapotok fognak érdekelni. Nem vizsgáljuk azt, hogy a rendszer hogyan kerül kezdeti állapotba, számunkra csak a kezdeti állapotból kiinduló rendszer müködés érdekes.

A 6. fejezet specifikációs nyelvének elemzésekor megemlitettük, hogy a nyelv lehetőséget teremt a protokoll tulajdonságok invariánsok formájában történő megfogalmazására. A pontosság érdekében most formálisan is definiáljuk a rendszer invariáns fogalmát.

Definició 8.2 : A (Q,R,N,P) átmenet rendszerében a Q állapot halmazon értelmezett I predikátum függvény invariáns (pontosabban q_0-invariáns) ha $\forall q \in Q$-ra $q_0 \xrightarrow{*} q$ esetén I(q) igaz értékü. □

A q_0-invariáns elnevezés arra utal, hogy a rendszer müködését a q_0 kezdeti állapotból inditottuk el. A q_0-invariáns pontositás szükségessége abból a tényből következik, hogy vannak olyan (Q,R,N,P) rendszerek, amelyek müködését ha a $q \neq q_0$ állapotból inditjuk el, akkor az I q_0-invariáns predikátum nem lesz q-invariáns. Ezzel kapcsolatos a következő tétel is.

Tétel 8.1 : Ha a (Q,R,N,P) rendszerben az I predikátum q_o-invariáns, akkor $\forall q \in Q$-ra $q_o \xrightarrow{*} q$ esetén az I q-invariáns.

Bizonyitás: Az állitás az elérhetőségi reláció alapvető tulajdonságaiból következik. Ha $\forall q \in Q$-ra $q_o \xrightarrow{*} q$ akkor $\forall q' \in Q$, $q \xrightarrow{*} q'$-re a $q_o \xrightarrow{*} q'$ teljesül a 8.1 definició 3. pontja miatt, igy I(q') igaz értékü. Tehát a $q \xrightarrow{*} q'$ esetén az I(q') igaz, ami az I q-invariáns voltát jelenti. □

Egy adott I predikátum invarianciájának ellenőrzése a 8.2 definició alapján a q_o kezdeti állapotból elérhető összes állapot "feltérképezését" követeli meg. Ezen az elvi alapon müködik a 8.3 fejezet állapotelérhetőségi analizis módszere. Mint majd később látni fogjuk a q_o állapotból elérhető q állapotok nagy száma jelentősen megnehezitheti az állapotelérhetőségi analizis tényleges megvalósitását, sőt egyes esetekben gyakorlatilag megakadályozhatja azt. Minden eddigi törekvésünk éppen az állapotok számának csökkentésére illetve könnyebb kezelésére irányult. Ez volt például az alapvető oka a (Q,R,N,P) rendszerek (π, α) reprezentánsa bevezetésének is. Szükség van tehát olyan invariancia ellenőrző módszerekre, amelyek nem követelik meg az összes q_o állapotból elérhető állapotok feltérképezését. Egy ilyen eljárást kinál a következő definició.

Definició 8.3 : A (Q,R,N,P) átmenet rendszer Q állapot halmazán értelmezett I predikátum függvény q_o-induktiv ($q_o \in Q$ a kezdő állapot) ha

1. $I(q_o)$ igaz és
2. $\forall q, q' \in Q$-ra az I(q) igaz és $q \rightarrow q'$ fennállása az I(q') = igaz kifejezést implikálja. □

A 8.2 és a 8.3 definiciókból triviálisan következik az alábbi tétel:

Tétel 8.2 : Ha a (Q,R,N,P) rendszerben az I predikátum q_o-induktiv akkor q_o-invariáns is. □

A 8.3 definició alapján egy I predikátum invariancia tulajdonságának vizsgálata a névvel ellátott átmenetek számával egyenlő számu vizsgálatot igényel. Meg kell tehát vizsgálni azt, hogy a (Q,R,N,P) rendszerben $\forall p \in P$ és $\forall n \in N$ esetén a

$$q \xrightarrow{p.n} q' \quad \text{és} \quad I(q) = \text{igaz}$$

kifejezésekből következik-e az I(q') = igaz állitás. Mivel a gyakorlatban a (Q,R,N,P) rendszerekben a p.n átmenetek száma jóval kisebb mint a q_0-ból elérhető állapotok száma ezért ebben az esetben nem jelentkezik az "állapotrobbanás" jelensége. Ezzel az előnnyel szemben azonban az egyes lépéseknél bonyolultabb tevékenységet kell elvégezni (az implikációt kell ellenőrizni) és ezt egy adott reprezentációnál nem mindig könnyü véghez vinni.

A protokoll deadlock szituációk formális megfogalmazása érdekében bevezetjük az AKTIV(q) predikátum függvényt.

Definició 8.4 : A (Q,R,N,P) rendszer Q állapotterén értelmezett AKTIV(q) predikátum függvény igaz értékü, ha az 5.2 definició E(q) halmaza nullánál több elemet tartalmaz, vagyis $q \in Q$-ra AKTIV(q) = igaz akkor és csakis akkor ha $\#E(q) \geq 1$. □

Definició 8.5 : A (Q,R,N,P) protokoll rendszer <u>deadlock mentes</u>, ha az AKTIV(q) predikátum függvény q_0-invariáns. □

Már a 8.1 fejezetben is megemlitettük azt, hogy nemcsak totális deadlock helyzetek vannak (a 8.5 definició ezekre vonatkozik), hanem deadlocknak nevezik azt a jelenséget is amikor a protokoll rendszernek csak bizonyos entitás csoportja jut holtpontra. Az ilyen holtpont helyzeteket a 8.5 definició nem fedi le, ezért egy "finomabb" deadlock definiciót kell adnunk. A pontosabb definició megalkotásánál Keller módszerét követjük, aki különleges, kitüntetett állapotátmenetek (key transitions) bevezetésével definiálja a deadlock mentesség fogalmát. Deadlock mentesnek tekinti a rendszert,

ha a predefiniált, kitüntetett átmenetek a rendszer müködése közben időről időre végrehajthatóvá válnak. A formális definició érdekében bevezetjük a p.n átmenethez tartozó $LIVE_{p.n}(q)$ predikátum függvényt.

Definició 8.6 : A (Q,R,N,P) átmenet rendszer p.n átmenetéhez rendelt ($p \in P$; $n \in N$) és a $q \in Q$ állapoton értelmezett $LIVE_{p.n}(q)$ predikátum függvény igaz értékü, ha $\exists q', q'' \in Q$ hogy $q \xrightarrow{*} q'$ és $q'' \in E(q')$ és

$$q' \xrightarrow{p.n} q''$$

teljesül. □

A definició áttekinthetőbben megfogalmazható a (Q,R,N,P) rendszer (π, α) reprezentációjában. Ebben a formalizmusban ugyanis a $LIVE_{p.n}(q)$ függvény igaz értékü ha $\exists q' \in Q$ hogy $q \xrightarrow{*} q'$ és $\pi_{p.n}(q') =$ igaz teljesülnek.

Definició 8.7 : A (Q,R,N,P) protokoll rendszer <u>deadlock mentes</u> ha $\forall p.n \in K$-ra (K a kitüntetett átmenetek halmaza) a $LIVE_{p.n}(q)$ predikátum függvény q_o-invariáns. □

Tétel 8.3 : Ha a (Q,R,N,P) rendszer a 8.7 definició értelmében deadlock mentes, akkor a 8.5 definició értelmében is az.

Bizonyitás: Indirekt módon feltételezve, hogy egy (Q,R,N,P) rendszerre a 8.7 definició igen de a 8.5 definició nem teljesül akkor $\exists q \in Q$ melyre $E(q) = \emptyset$. Ekkor viszont $\forall p.n \in K$-ra a $LIVE_{p.n}(q) =$ hamis ami nyilvánvalóan ellentmondásban van a kiindulásunkkal. □

A 8.7 definició a kitüntetett átmenetek megválasztásáról nem határozott. A deadlock mentesség vizsgálatakor a tervező szabadon választhatja meg azokat a kitüntetett átmeneteket, amelyek rendszeres végrehajtása nézete szerint a protokoll müködés elengedhetetlen velejárója. A 8.7 definicióval kapcsolatos az a másik megjegyzésünk is, hogy a 9. fejezet verifikáló rendszerében a deadlock mentesség fogal-

mát a 8.7 definicióhoz képest kibővitve használjuk, ugyanis nem tekintjük holtpontnak azokat a 8.7 definició szerint holtpont helyzeteket, amelyekben a WAIT predikátum (6.4 fejezet) igaz értékü.

A liveness tulajdonság formális megfogalmazása pontosan követi a 8.1 fejezet szöveges leirását.

Definició 8.8 : Az ELÉRHETŐ$_q$,(q) predikátum igaz értéket vesz fel minden olyan $q \in Q$-ra melyre $q \xrightarrow{*} q'$ teljesül. □

Definició 8.9 : A (Q,R,N,P) átmenet rendszer liveness tulajdonságu ha teljesül a

(ELÉRHETŐ$_q$ q_o-invariáns) q_o-invariáns. □

A szokatlan formáju kifejezésben a zárójeles rész azt fejezi ki, hogy egy rögzitett $q \in Q$ elérhető minden a $q_o \in Q$ kezdőállapotból elérhető q állapotból, mig a külső q_o-invariancia a zárójelben rögzitett q-ra vonatkozik. A liveness tulajdonság ellenőrzését gyakran megkönnyiti az, hogy a protokollok jelentős részében van egy kitüntetett un. "home" állapot(q_H). Ezen állapot minden más q_o-ból elérhető állapotból elérhető, vagyis

ELÉRHETŐ$_{q_H}$ q_o-invariáns.

A liveness tulajdonság ellenőrzése ekkor abból áll, hogy a q_H-ból elérhető állapotok halmazát össze kell vetni a q_o-ból elérhető állapotok halmazával. Ha a két halmaz megegyezik, akkor a rendszer liveness tulajdonságu.

A progress sajátság formalizálásának nehézségeiről a 8.1 fejezetben már szóltunk. Itt most nem teszünk kisérletet a pontosabb definició megadására.

A deadlock mentesség duális problémája a terminálódás kérdése. Amig a deadlock mentesség azt fejezi ki, hogy a protokoll rendszer sohasem terminálódik, addig vannak olyan protokollok, amelyeknek mindig terminálódni kell. A köteles-

ző terminálódás megfogalmazásához a q_0-invarianciánál "gyengébb" invariancia fogalom is elegendő.

Definició 8.10 : A (Q,R,N,P) átmenet rendszer Q állapotterén értelmezett I predikátum függvény q_0-szubinvariáns, ha van olyan a q_0-ból kiinduló rendszer müködés (5.7 definició) amelynek minden $q \in Q$ állapotára az I(q) igaz értékü. □

Definició 8.11 : A (Q,R,N,P) rendszer mindig terminálódik, ha nincs olyan rendszer müködés, amely minden q állapotára az AKTIV(q) q_0-szubinvariáns. □

Ezen fejezetben elmondottak alapján megállapitható, hogy az invariánsok segitségével egyes intuitiv protokoll tulajdonságok könnyen, mások viszont csak nagy nehézségek árán formalizálhatók. Bizonyos definiciók az adott tulajdonság vizsgálatát lehetővétévő gyakorlatban alkalmazható módszereket sugallnak, mig mások csak elméleti értékük.

8.3 A bővitett állapotelérhetőségi analizis

A bővitett állapotelérhetőségi analizis tetszőleges predikátumok q_0-invariáns illetve q_0-szubinvariáns voltának ellenőrzését lehetővé tévő verifikációs módszer. A 8.2 fehezetben formálisan megfogalmaztuk az alapvető protokoll tulajdonságokat, bizonyos predikátum függvények invarianciájaként kifejezve azokat. A bővitett elérhetőségi analizis tehát alkalmas ezen protokoll tulajdonságok felderitésére. A 9. fejezet számitógéppel támogatott verifikáló rendszerében lehetőség van egyéb, például a protokollok funkcionális helyességét biztositó predikátumok q_0-invarianciájának ellenőrzésére is.

Az 5.11 definició meghatározta a protokollok un. totális specifikációját. Elemeztük azokat az okokat, amelyek

miatt a gyakorlatban ez a specifikációs módszer nem alkalmazható. Helyette a protokollok (π,α) reprezentációjának megadását javasoltuk. A tervező azonban egy (π,α) reprezentáció megalkotásával olyan specifikációs módszert is létrehozhat, amely nem garantálja azt, hogy a formalizmus keretén belül megfogalmazott protokollok helyesek lesznek, rendelkezni fognak az alapvető elvárt tulajdonságokkal (deadlock mentesség, liveness, progress tulajdonság stb.). Jelen munkában a (π,α) reprezentáció egy programozási nyelvi kifejtésére tettünk javaslatot. A nyelv szintaktikai, szemantikai szerkezetei bár nagy számu protokoll hiba elkerülését biztositják, de például nem tudják garantálni azt, hogy csak deadlock mentes protokollt lehet specifikálni ezen a nyelven. Egyszerübben fogalmazva a jelen protokoll specifikációs nyelv tulságosan tág teret kinál a felhasználójának, mivel hibás specifikálást is megenged. (A jövő utja valószinüleg éppen az olyan szabályok kidolgozása és nyelvi konstrukciókba "ültetése" lesz, amelyek betartása automatikusan helyes (pl. deadlock mentes) protokollt fog eredményezni. Az ilyen konstruktiv tervezési szabályok kialakitására még csak néhány kisérleti lépés történt. Reméljük, hogy ezen verifikációs módszerrel nyert tapasztalatok hozzá fognak járulni a protokoll rendszerek sajátosságainak mélyebb megértéséhez és ezen keresztül talán a fennt emlitett konstruktiv tervezési szabályok kialakitásához is.)

Az alább bemutatott elérhetőségi analizis lényegében a (π,α) reprezentációból rekonstruálni próbálja a (Q,R,N,P) átmenet rendszert, ami a Q és R halmazok elemeinek, az állapotok és a köztük lévő relációk feltérképezéséből áll. A (Q,R,N,P) rekonstruálás a (Q,R,N,P) grafikus képének, az un. eléhetőségi gráfnak a felépitését jelenti. Összefüggést találva az elérhetőségi gráf strukturális (geometriai) tulajdonságai és a 8.2 fejezetben felsorolt protokoll tulajdonságok között, a 3.2 fejezet első két verifikációs feladat tipusa az elérhetőségi gráf strukturális vizsgálatára vezethető vissza. A következőkben definiálni fogjuk a (π,α)

nyelvi reprezentánsa esetén a rendszer állapot fogalmát, majd megadjuk a (Q,R,N,P) grafikus képét (pontosabban azt a leképezést, amely a (Q,R,N,P)-ből előállitja az elérhetőségi gráfot) és végül bemutatjuk a protokoll tulajdonságok és az elérhetőségi gráf geometriai strukturája közötti kapcsolatot.

A ($\mathcal{R}$,$\prec$) reprezentációban a rendszer állapotának definiálása a REP^{-1} inverz függvény (5. fejezet) megadásával egyenértékü. A specifikációs nyelv a protokollt entitások összekapcsolt rendszerének tekinti. A rendszer (globális) állapot tehát az entitások állapotainak függvénye:

$$q = REP^{-1}(e_1, e_2, \ldots e_j, \ldots e_n)$$

ahol e_j a j-edik entitás állapota, n pedig a rendszer entitásainak száma. Az entitás állapota az állapotváltozói és interfészváltozói állapotától, vagyis értékeitől függ:

$$e_j = REP_e^{-1}(a_{j1}, a_{j2}, \ldots a_{jk}, \ldots a_{js_j} \; i_{j1}, i_{j2}, \ldots i_{jl}, \ldots i_{jt_j})$$

ahol s_j a j-edik entitás állapotváltozóinak, t_j pedig interfészváltozóinak száma. Mivel az egyes állapot- illetve interfészváltozók összetett adattipusok is lehetnek, ezért tovább kellene bontanunk az a_{jk} és i_{jl} állapotokat. Ezt nem tesszük meg, de utalunk arra, hogy például két csomagsor (QueueOfPacket) tipusu változó állapota akkor egyezik meg, ha ugyanazok a csomagok ugyanolyan sorrendben helyezkednek el bennük. A REP^{-1} és REP_e^{-1} függvények konkrét alakját a legegyszerübb skalár-vektor függvények (skalár értékü vektor változós függvények) közül választjuk.

Definició 8.12 : Az entitások állapota az állapot- és interfészváltozók állapotainak vektora:

$$e_j = \langle a_{j1}, a_{j2}, \ldots a_{js_j}, i_{j1}, i_{j2}, \ldots i_{jt_j} \rangle \qquad (1)$$

a j-edik entitás állapota. □

Definició 8.13 : A protokoll rendszer (globális) állapota a résztvevő entitások állapotaiból álló vektor:

$$q = \langle e_1, e_2, \ldots e_j, \ldots e_n \rangle \qquad (2)$$

□

Az (1) kifejezést a (2)-be helyettesitve a

$$q = \langle a_{11}, \ldots a_{ns_n}, i_{11}, \ldots i_{nt_n} \rangle \qquad (3)$$

állapot definiciót kapjuk. A (3) kifejezés redundáns, mivel az összekapcsolt interfészváltozók értéke természetszerüleg azonos (hiszen valójában közös, "osztott" változóról van szó), igy állapotaik is megegyeznek. Ez nem befolyásolja a (3) használhatóságát, de esetleg memória takarékossági szempontból a számitógépes megvalósitásnál figyelembe vehető optimalizálási lehetőséget kinál.

A (Q,R,N,P) átmenet rendszer grafikus képe a $G(\mathcal{Z}, \mathcal{E}, \mathcal{Y})$ irányitott gráf, amelyet elérhetőségi gráfnak nevezünk. A G gráfban $\mathcal{Z}$ a gráf csucspontjainak, $\mathcal{E}$ az irányitott éleknek és $\mathcal{Y}$ olyan leképezéseknek a halmaza, amely az irányitott éleket csucspontpárokhoz rendeli. ($\mathcal{E} \longrightarrow \mathcal{Z} \times \mathcal{Z}$) A protokoll rendszer globális állapotait (Q) a gráf csucspontjainak, az állapotátmeneteket a gráf irányitott éleinek megfeleltetve, valamint megkövetelve azt, hogy

$$q \xrightarrow{p.n} q'$$

esetén a q-nak megfeleltetett csomópontból a p.n átmenetnek megfeleltetett (p.n-nel cimkézett) irányitott él vezessen a q'-höz hozzárendelt csomópontba, megadtuk a (Q,R,N,P) rendszer elérhetőségi gráfra történő leképezését.

A (π, α) reprezentáció ismeretében az elérhetőségi gráf felépitésére a következő algoritmust adhatjuk. Egy q rendszer állapothoz rendelt gráfpontból induljon ki annyi irányitott él, amennyi állapotátmenet végrehajtható (5.3 definició) az adott állapotban. Cimkézzük ezeket az éleket a végrehajtható átmenetek (p.n) nevével. A p.n névvel cimkézett él a q állapotu rendszer p.n állapotátmenetének végrehajtása utáni állapotához hozzárendelt gráf-

pontba vezessen. A q_0 kezdeti állapotból inditott rendszer elérhetőségi gráfját felépitő algoritmusunk megáll akkor, ha minden a q_0-ból elérhető állapotra fennáll az előző mondat állitása. Elképzelhető, hogy egy protokoll rendszer elérhetőségi gráfja végtelen nagyságu (hibás protokoll) ekkor algoritmusunk is végtelen ideig müködik. Az ilyen esetek kiszürése a számitógépes megvalósitásban az ember közremüködését igényeli.

Az elérhetőségi gráf geometriai (strukturális) tulajdonságai és a protokoll tulajdonságok közötti összefüggés a 8.2 fejezet megfelelő definiciói alapján könnyen belátható.

Ha a G elérhetőségi gráfban van olyan csomópont, amelyből nem vezet ki irányitott él, akkor a csomóponthoz tartozó q állapot a protokoll deadlock állapota. Deadlock mentes tehát a protokoll ha az elérhetőségi gráf minden pontjából vezet irányitott él valamely másik csomópontba.

A protokoll liveness tulajdonságu ha az elérhetőségi gráfja erősen összefüggő. (Egy irányitott gráf erős összefüggőségi tulajdonsága azt fejezi ki, hogy bármely csomópontból vezet irányitott élsorozat bármely másik csomópontba [Birkhoff és m. 74].) Az irányitott gráfok erős összefüggőségének meghatározására számos az irodalomban is publikált algoritmus adható meg (pl. [Aho és m. 82]). Két megjegyzést kell tennünk.

Gyakorlati szempontból célszerü, ha a deadlock mentesség vizsgálata megelőzi a liveness tulajdonság vizsgálatát. Ez annál is kézenfekvőbb, mivel a deadlock állapotok az elérhetőségi gráf épitése közben is felderithetők. Ha a protokoll nem deadlock mentes akkor biztos, hogy nem liveness tulajdonságu és ilyenkor felesleges elvégezni a nem tul egyszerü erős összefüggőségi vizsgálatot.

Mint azt már a 8.2 fejezetben is megemlitettük, ha

van a protokollnak un. "home" állapota (q_H) és $q_H = q_o$ teljesül, akkor a protokoll liveness tulajdonságu. Ezen az alapon jelentősen egyszerüsödhet a liveness tulajdonság meglétének vizsgálata. Ha a protokoll deadlock mentes és a q_o-ból induló minden ciklus (lásd. később) a q_o-ban záródik, akkor a $q_o = q_H$ és igy a protokoll liveness tulajdonságu.

A protokoll ciklusait a G irányitott gráf irányitott hurkainak feltérképezésével kaphatjuk meg. Az egyes ciklusok szükséges vagy felesleges voltának meghatározása (a progress tulajdonság vizsgálata) szintén az ember feladata.

A különféle egyéb q_o-invariánsok formájában megfogalmazott tulajdonságok elemzése az invariánsok az elérhetőségi gráf csomópontjaikénti kiértékelésével elvégezhető, az invariáns sértések ilyen módon felderithetők. Egy adott predikátum q_o-invariáns, ha az elérhetőségi gráf minden csomópontjához rendelt rendszer állapotra igaz értéket szolgáltat.

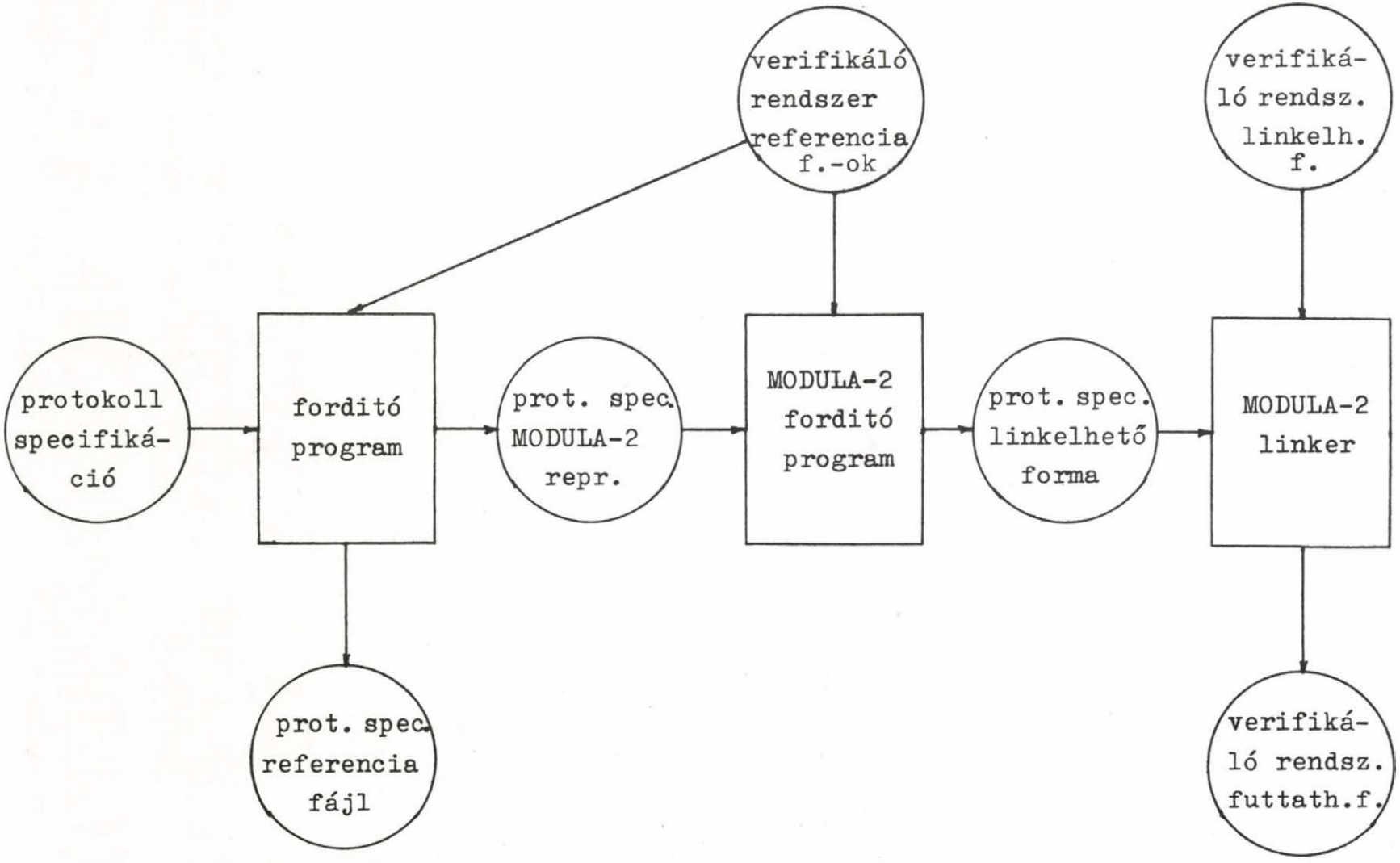
Egy predikátum q_o-szubinvariáns ha van az elérhetőségi gráfban egy olyan irányitott élekből álló q_o-ból induló hurok, amelyben szereplő összes csomóponthoz rendelt rendszer állapotra a predikátum függvény igaz értéket ad.

ban olyan egyszerüek, hogy az manuálisan is könnyen elvégezhető. (Jelen pillanatban a precompiler hiányában mi is ezt az utat követjük.) A 9.1.4 fejezetben konkrét példa kapcsán részletesen tárgyalni fogjuk a protokoll specifikáció MODULA-2 nyelvü reprezentációját. A MODULA-2 forditóprogrammal leforditott linkelhető formából és a verifikáló rendszer többi moduljának LNK kiterjesztésü fájljaiból a linker alakitja ki a korábban már emlitett futtatható formát.

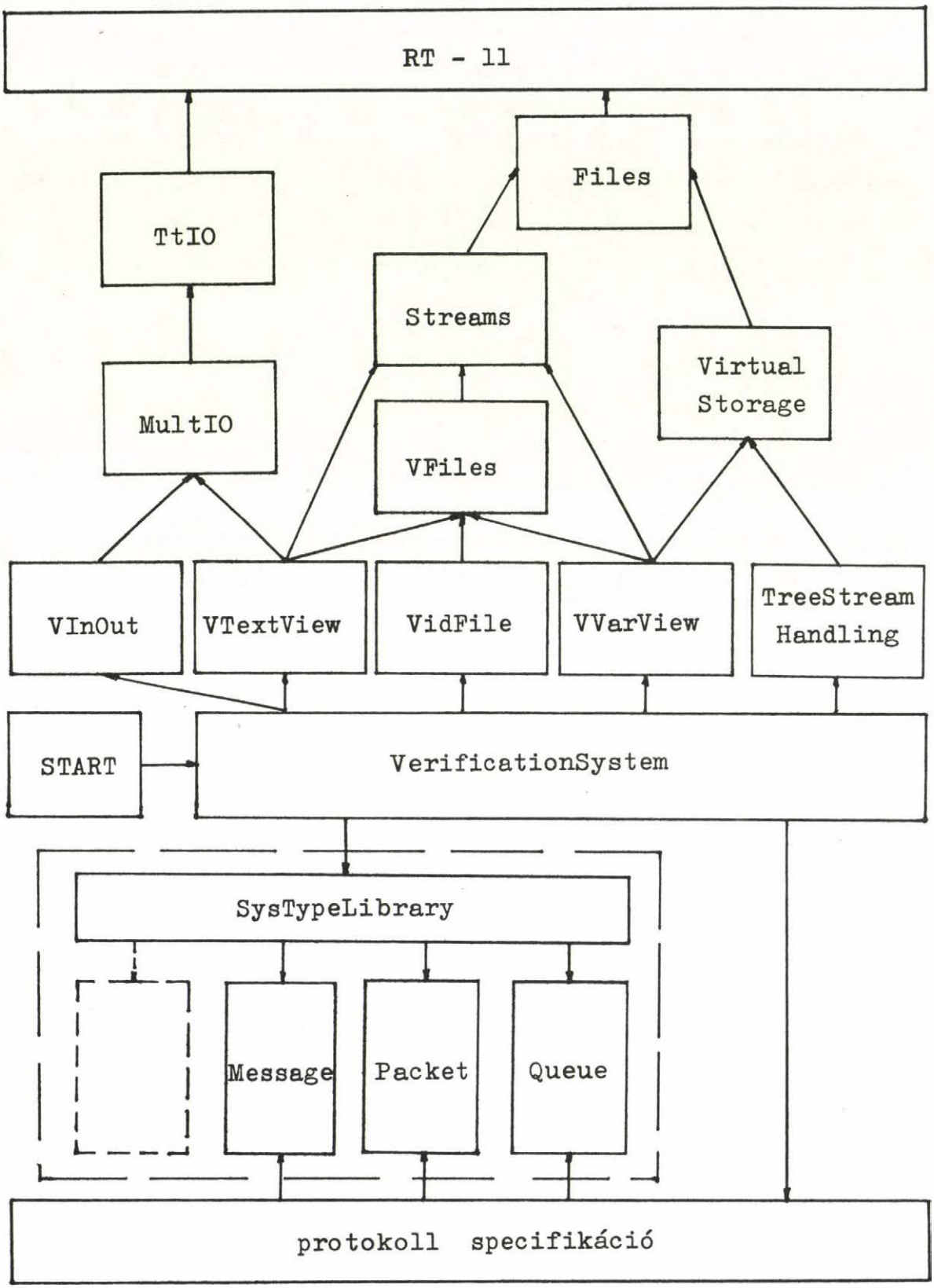
9.1 A verifikáló rendszer felépitése

A 9.2 ábra a verifikáló rendszer strukturáját mutatja be, feltüntetve a fontosabb modulokat és hozzáférési jogokat.

Az ábra alján a protokoll specifikációt tartalmazó modul helyezkedik el. Hozzáférési jogokkal rendelkezik az absztrakt adattipus könyvtár (szaggatott vonallal bekeretezve) egyes adattipusait megvalósitó modulokhoz. A MODULA-2 nyelv megköveteli, hogy a modulok export és import listái statikusak, vagyis már forditási időben ismertek legyenek. Ez azt jelenti, hogy a főmodulnak (VerificationSystem) ismernie kell a protokoll specifikációját tartalmazó modul nevét és az onnan importált változók, eljárások neveit is. Ez ellentmondásban van azzal az igénnyel, hogy egy ujabb protokoll verifikálásánál ne kelljen megváltoztatni a verifikáló rendszer állandó moduljait. Az ellentmondás feloldására kinálkozó lehetőségek közül mi a legegyszerübbet választottuk, rögzitettük a protokoll specifikációjának MODULA-2 reprezentánsát tartalmazó modul nevét és a verifikáló rendszer állandó moduljai felé mutatott interfészét. A főmodul felől tehát rögzitett modul és eljárás nevek láthatók.



9.1 ábra A verifikáló rendszer használatának vázlata



9.2 ábra A protokoll verifikáló rendszer felépitése

Az absztrakt adattipus könyvtárban az egyes adattipusok külön-külön modulokban helyezkednek el. Közülük egyet (QueueModule) rendszer célokra is felhasználunk, igy az rögzitett. A könyvtár bővitését szaggatott vonalakkal illusztráltuk. Az adattipus könyvtár verifikáló rendszer felé mutatott egységes interfészéről gondoskodik a SysTypeLibrary modul.

A rendszer legnagyobb és egyben legfontosabb modulja a VerificationSystem modul, amely az elérhetőségi gráf generáló algoritmus megvalósitását tartalmazza. A háttértárban elhelyezett elérhetőségi gráf épitését könnyiti meg a TreeStreamHandling modul, amely a háttértár memóriakénti viselkedését biztositó VirtualStorage modulon keresztül kapcsolódik az RT-11 fájl rendszer alatt közvetlenül elhelyezkedő Files modulhoz. A Files az RT-11 fájlok nyitását, zárását, uj fájlok létrehozását, régi fájlok törlését végzi. Alapszolgáltatásként egy kiválasztott fájl, kiválasztott blokkjához enged hozzáférést, blokkirás, blokkolvasás eljárások formájában. Ezen alapszolgáltatásra épülve a VirtualStorage modul homogén, egy fájlban elhelyezkedő memóriát hoz létre. Ebbe a memóriába szavakat (vagy bájtokat) lehet irni és onnan kiolvasni a cimzésnek megfelelően. A Streams standard modul szigoruan soros hozzáférésü szó (vagy bájt) sorozatokat (sequence) hoz létre. A VFiles modul gondoskodik a verifikáló rendszer müködéséhez szükséges fájlok kezeléséről, nyilvántartásáról, nyitásáról, zárásáról.

A fájl kezeléshez hasonlóan a képernyő input-output kezelést is rétegesen egymásra épülő modulok sorozatával oldjuk meg. Az egyes modulok a hierarchikus rendszerben alacsonyabb szinten elhelyezkedő modulok szolgáltatásaira (eljárásaira)épitve egyre magasabb szintü, absztraktabb szolgáltatásokat nyujtanak. Törekedtünk arra, hogy ezt az elvet ne csak modul szinten, de a modulok belsejében lévő eljárások szintjén is intenziven alkalmazzuk, eredményképpen pedig jól áttekinthető, strukturált programot kapjunk. A TtIO modul

még csak egy karakter terminál input-outputját oldja meg, a MultIO modul viszont már egész számok,(integer, cardinal) sorok ki-be vitelére képes. A VInOut modul a verifikációs rendszer központi parancs értelmezője. Segitségével történik a képernyő különféle tartományainak (domain) inicializálása, kezdeti feliratozása. A VTextView modul az elérhetőségi analizis külső parancsra történő felfüggesztésekor a protokoll specifikáció képernyőn történő megjelenitésére, az abban történő előre, hátra lapozásra szolgál. Hasonló funkciót lát el a VVarView, amely a specifikáció változói értékeinek megjelenitését végzi. A VidFile modul a protokoll specifikáció azonositóiról referencia információkat tartalmazó segédfájl feldolgozását könnyiti meg. A START modul a rendszer inditásáért felel.

A 9.2 ábrán az áttekinthetőség kedvéért nem tüntettük fel azokat a standard rendszer modulokat, amelyek nem lényegbe vágó funkciókat hordoznak. Feladatuk csupán programozás technikai egyszerüsités, az ábráról történő lemaradásuk a megértést nem korlátozza. Ugyancsak az egyszerüség kedvéért az ábra nem tartalmaz néhány olyan hozzáférési jogot, amely a hierarchikus szerkezetet megkerülve hatékonyabb (gyorsabb) futást biztosit.

9.1.1 Az elérhetőségi gráf reprezentációja

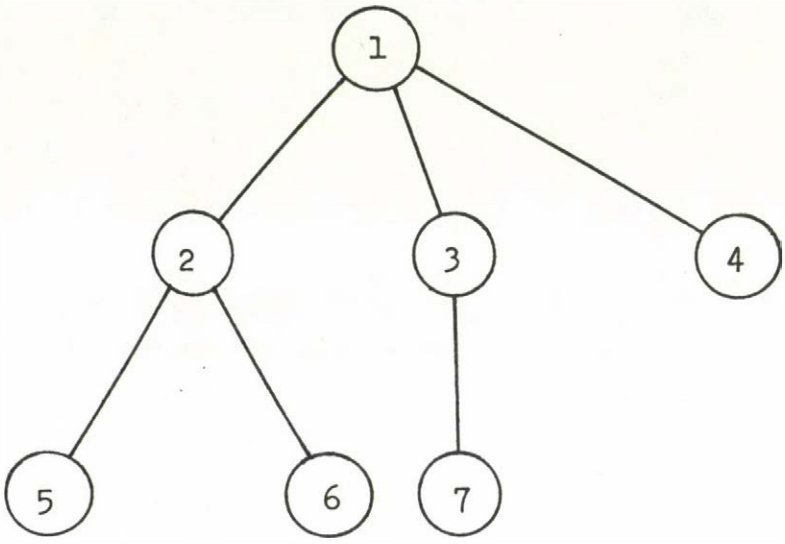
A protokoll verifikáló rendszer tervezésének kiinduló pontja azok az irodalomban publikált tapasztalatok voltak (magyarországi tapasztalatokról mindezidáig nem beszélhettünk), amelyek szerint a protokoll verifikálás még számitógépes támogatás esetén is hosszu folyamat. Interaktiv rendszert használva a verifikálás időigénye jelentősen meghaladja(hatja) az egy-egy alkalommal képernyő előtt tölthető időt, több munkanapra, esetleg munkahétre is kiterjedhet. Ebből

az következik, hogy a rendszernek lehetőséget kell nyujtania a verifikálás menetének megszakitására, majd későbbi folytatására. Két munka alkalom között a verifikálás menetéről szóló információkat megőrzésre a háttértárolón (mágneslemezen) kell elhelyezni. A verifikálás menetéről szóló eme információk legsüritettebb formája éppen az elérhetőségi gráf, igy célszerüen ezt fogjuk eltárolni.

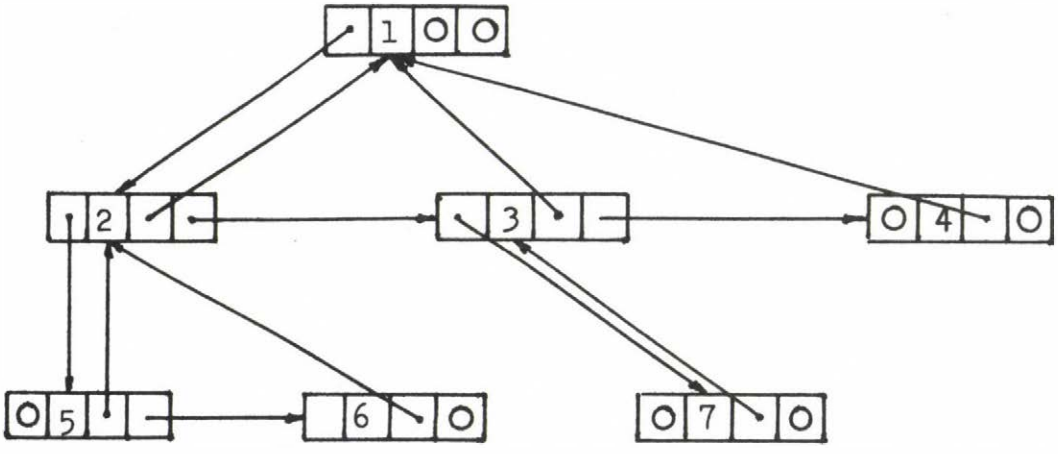
Az elérhetőségi gráf méretének becsléséhez a gráf várható átlagos állapot számát kell figyelembe venni. Az irodalomban beszámolnak arról, hogy közepes méretü protokoll esetén is a globális állapotok száma ezres nagyságrendü, véges állapotu gépes leirási módot alkalmazva. Specifikációs nyelvünk jelentősen csökkentette a globális állapotok számát, de még igy is ezres nagyságrendre készülünk fel. Egy globális állapot tárolásához szükséges helyigény a 8.13 definició alapján, a TPA-11/40 16 bites szavait alapulvéve mintegy 50-100 szóra becsülhető. Ráadásul ez a helyigény állapotról állapotra változhat, hiszen a specifikációban dinamikus adatstrukturák (pl. várakozási sorok) szerepelnek, amelyek mérete (tartalmuk) futás közben módosul. (A 7. fejezet Bitalternáló protokoll globális állapotának helyigénye minimálisan 17 szó, feltételezve, hogy a sor tipusu változók mindegyike üres (nulla hosszuságu sorok) és nem számolva a gráf csucspontjai között lévő élek leirásához szükséges mutatók által elfoglalt helyet.) A MODULA-2-ben is szereplő adattipusok (Integer stb.) tárolásánál a MODULA-2 rendszer implementációját követjük (pl. egy szavas Integer). A csomag (Packet) és üzenet (Message) tipusu változókat a 9.1.3 fejezet alatt részletezett okok miatt egy szóban, a különféle sor tipusu változókat az aktuális hosszuknak megfelelő számu plusz egy, a sorhosszuságot tartalmazó szóban tároljuk.

A G elérhetőségi gráf (8.3 fejezet) többágu (nem bináris)fával reprezentálható, ugyanis egy globális állapotból, éppen a rendszer nemdeterminisztikussága miatt több végrehajtható állapotátmenet is leképzelhető. A fa csucsait, a

globális állapotokat mágneslemezen tároljuk. A fa éleinek leirására "pointereket" (két szavas mágneslemez tárcimeket (a programban Position)) használunk. A többágu fát a háttértárolón bináris faként ábrázoljuk, igy az aktuális állapotból kiinduló mutatók száma nem függ a gráf csucspontjaiból kiinduló élek számától. A többágu fa binárisként való ábrázolását illusztrálja a 9.3 és 9.4 ábra.



9.3 ábra Az elérhetőségi gráf mint többágu fa



9.4 ábra Az elérhetőségi gráf bináris fa ábrázolása

A 9.4 ábrán a fa könnyebb bejárása érdekében a megfelelő gyökérelemekre visszamutató pointereket is alkalmaztunk. A 9.5 ábra részletesen bemutatja az elérhetőségi gráf csomópontjának felépitését MODULA-2 nyelven.

```
TYPE
    ErrorType = (good,deadlock,beforedeadlock,
                 wait,invarianterror,cyclefound);

    TransitionSet = ARRAY[0..ENTITYNUM-1] OF BITSET;

    Node = RECORD
             father, sublist, nextnode       : Position;
             error                           : ErrorType;
             tset                            : TransitionSet;
             entitynumber,transitionnumber   : CARDINAL;
           END;
```

9.5 ábra Az elérhetőségi gráf csomópontjának felépitése

A csomópont a megfelelő mutatók (pointerek) mellett az állapotot minősitő (error) mezőt, valamint a végrehajtható de még nem "bejárt" állapotátmenetekre utaló (tset) mezőt is tartalmazza. A rekord két végső mezője az utoljára végrehajtott állapotátmenet adatait hordozza.

Az elérhetőségi gráf mágneslemezen történő tárolását teszit lehetővé a VirtualStorage modul, amelynek definiciós modulját alább közöljük.

A definiciós modul a VSTORAGE, a Position tipusokat és a rajtuk értelmezett müveleteket exportálja. A CreateV-Storage eljárás létrehozza a szavakból vagy bájtokból (char) álló, memóriaként viselkedő adatstrukturát. Ennek egy része a mágneslemezen lévő valamely megnyitott fájlban, másik része a memóriában helyezkedik el. A felhasználó szemszögéből azonban ez a kettéosztás rejtve marad. Olyan homogén cimtartományu szó vagy bájt memória jön létre, amelynek hosszuságát csak a mágneslemez nagysága korlátozza. A KillVStorage és az EndWrite eljárások a VSTORAGE memória elmentését végzik.

```
(*************************************************************
*                                                           *
*              Protocol Verification System                 *
*                                                           *
*                 Virtual Storage Module                    *
*                                                           *
*************************************************************)

DEFINITION MODULE VirtualStorage;

FROM SYSTEM IMPORT WORD;

EXPORT QUALIFIED VSTORAGE, Position, CreateVStorage,
  KillVStorage, Reset, EndWrite, WriteWord, WriteChar,
  ReadWord, ReadChar, GetPos, SetPos, IncPos, DecPos,
  EqlPos, SetNullPos;

  TYPE
    VSTORAGE;
    Position=RECORD highpos, lowpos:CARDINAL END;

  PROCEDURE CreateVStorage(VAR v       :VSTORAGE;
                               filenum:CARDINAL;
                               ws      :BOOLEAN);
    (* ws means: 'v' is a word, not a byte storage *)

  PROCEDURE KillVStorage(VAR v:VSTORAGE; closefile:BOOLEAN);

  PROCEDURE Reset(v:VSTORAGE);

  PROCEDURE EndWrite(v:VSTORAGE);


  PROCEDURE WriteWord(v:VSTORAGE; w:WORD);

  PROCEDURE WriteChar(v:VSTORAGE; ch:CHAR);

  PROCEDURE ReadWord(v:VSTORAGE; VAR w:WORD);

  PROCEDURE ReadChar(v:VSTORAGE; VAR ch:CHAR);


  PROCEDURE GetPos(v:VSTORAGE; VAR p:Position);

  PROCEDURE SetPos(s:VSTORAGE; p:Position);

  PROCEDURE IncPos(VAR p:Position; with:CARDINAL);

  PROCEDURE DecPos(VAR p:Position; with:CARDINAL);

  PROCEDURE EqlPos(p1,p2:Position):BOOLEAN;

  PROCEDURE SetNullPos(VAR p:Position);

END VirtualStorage.
```

A WriteWord (WriteChar), ReadWord (ReadChar) müveletekkel az aktuáli cimről (Position) olvasni vagy oda irni lehet. A GetPos az aktuális VSTORAGE memória cim lekérdezését, a SetPos a megadott cimre való ugrást végzi. Az IncPos, DecPos eljárások segitségével a cimet adott értékkel növelni vagy csökkenteni lehet. Aze EqlPos igaz értékü, ha a két cim megegyezik. A SetNullPos a p Position tipusu változónak Null értéket ad, amely kitüntetett érték, a szokásos NIL pointer VSTORAGE memóriabeli megfelelője.

9.1.2 Input - output kezelés

A protokoll rendszer müködésének külső megfigyelhetőségét az állapotátmenetek nevének (5.4 definició) bevezetésével biztositottuk. A verifikáló rendszer felhasználójának a rendszer müködés (5.9 definició) tehát állapotátmenetek neveinek sorozata. A program rendszer (verifikáló rendszer) belső müködése során az állapotátmeneteket számokkal (az entitás és azon belüli átmenet sorszámával) azonositja. Jelenlegi implementációban az entitásonkénti átmenetek számát maximálisan 16-ra korlátoztuk, mivel ugy gondoltuk, hogy tizegynéhány átmenet még viszonylag könnyen áttekinthető. Nagyobb entitásonkénti átmenetszám igény kielégitése sem ütközik áthághatatlan akadályokba, néhány konstans kicserélését és a verifikáló rendszer bizonyos moduljainak ujraforditását követeli meg.

Az átmenetek neveit a protokoll specifikáció MODULA-2 nyelvü reprezentációja nem tartalmazza. Helyette egy külön referencia fájlba gyüjtöttük a nyomkövetéshez szükséges információkat. Az IDF kiterjesztésü referencia fájl kötött formában rögziti az entitások számát, az entitás neveket, az entitások állapot- és interfészváltozói nevét, tipusát

és mindezeknek a specifikációs fájlbeli deklarációjának sorszámát. A verifikáló rendszer ezen referencia fájl feldolgozásával jut az entitások és az állapotváltozások neveinek birtokába.

A nevek és az egyéb információk képernyőn történő megjelenitését végzi a MultIO modul.

```
(*************************************************************
*                                                           *
*           Multidomain display input/output                *
*                                                           *
*                    (VT 52 , VT 100)                       *
*                                                           *
*************************************************************)

DEFINITION MODULE MultIO;      (* Laszlo Kovacs 05-Febr-82 *)

  FROM SYSTEM IMPORT WORD;
  FROM Files IMPORT FileName;
  EXPORT QUALIFIED
    LINELENGTH, Line, OpenInputFile, OpenOutputFile,
    CloseInputFile, CloseOutputFile, CreateDomain,
    MoveCursorHome, ClearDomain, MoveCursor, Read, ReadInt,
    ReadString, Write, WriteString, ShowString, WriteInt,
    WriteCard, WriteOct, WriteHex, Writeln, SwitchDomain;

  CONST LINELENGTH = 80;
  TYPE  Line = ARRAY [0..LINELENGTH-1] OF CHAR;

  PROCEDURE OpenInputFile(input    :FileName;
                          dialogue:BOOLEAN;
                          domain   :INTEGER);
  PROCEDURE OpenOutputFile(output    :FileName;
                           overwrite:BOOLEAN;
                           dialogue :BOOLEAN;
                           domain   :INTEGER);
  PROCEDURE CloseInputFile;
  PROCEDURE CloseOutputFile;

  PROCEDURE CreateDomain(domain                    :INTEGER;
                         topline, topcolumn        :INTEGER;
                         bottomline, bottomcolumn:INTEGER;
                         pagemode                  :BOOLEAN);
   (* domain is positiv integer, 0 domain is a predefined
      standard one for the orisinal display screen *)
  PROCEDURE MoveCursorHome(domain:INTEGER);
  PROCEDURE ClearDomain(domain:INTEGER);
  PROCEDURE SwitchDomain(domain:INTEGER);
  PROCEDURE MoveCursor(line, column:INTEGER);
    (* absolut cursor address,
       1 <= line <= 24, 1 <= column <= 80 *)
```

```
PROCEDURE Read(VAR ch:CHAR; echo:BOOLEAN; domain:INTEGER);
PROCEDURE ReadInt(VAR x       :INTEGER;
                  VAR nextChar:CHAR;
                  VAR isNum   :BOOLEAN;
                      echo    :BOOLEAN;
                      domain  :INTEGER);
  (*skip blanks and control characters; if a sequence of
    digits (possibly preceded by a sign) follows, read
    it as a decimal integer; isNum indicates whether a
    number was read, nextChar gives the last character
    read, i.e. the one following the sequence of digits;
    OC signals the end of the input stream. No tests
    for overflow are made.*)
PROCEDURE ReadString(VAR s      :ARRAY OF CHAR;
                         echo   :BOOLEAN;
                         domain:INTEGER);

PROCEDURE Write(ch:CHAR; domain:INTEGER);
PROCEDURE WriteString(s:ARRAY OF  CHAR; domain:INTEGER);
PROCEDURE ShowString(s      :ARRAY OF  CHAR;
                     cr     :BOOLEAN;
                     domain:INTEGER);
PROCEDURE WriteInt(x:INTEGER; n:CARDINAL; domain:INTEGER);

PROCEDURE WriteCard(x, n:CARDINAL; domain:INTEGER);
PROCEDURE WriteOct(w:WORD; n:CARDINAL; domain:INTEGER);
PROCEDURE WriteHex(w:WORD; n:CARDINAL; domain:INTEGER);
PROCEDURE Writeln(domain:INTEGER);

END MultIO.
```

A CreateDomain eljárás segitségével a felhasználó független tartományokat (domains) hozhat létre a képernyőn. Ezek a tartományok önálló képernyőként funkcionálnak. Két alaptipus, a lapiró (page) és a görgetett (scroll) üzemmódu tartomány definiálható. Minden tartománynak saját "home" poziciója van, amelybe a lokális tartományi cursort a MoveCursorHome eljárással lehet mozgatni. A ClearDomain a tartomány törlését végzi. Az egyes tartományokból karaktert, karakter füzért illetve integert lehet beolvasni (Read, ReadString, ReadInt) illetve oda kiirni (Write, WriteString, WriteInt). A Writeln eljárás az adott tartomány kocsi vissza soremelés funkcióját látja el. A modul számos egyéb, itt nem tárgyalt szolgáltatást is nyujt, amelyeket a verifikáló rendszerben nem használunk ki. Ezen többlet szolgálta-

tások megléte a modul általánosabb célu felhasználási területével függ össze.

9.1.3 Az absztrakt adattipus könyvtár

A 6.1 fejezetben már ismertettük az absztrakt adattipusok rendszerünkben betöltött szerepét. Jelen fejezetben az absztrakt adattipusok megvalósitását és könyvtárba foglalását tárgyaljuk.

Az absztrakt adattipusok adatstrukturából és a rajtuk értelmezett müveletekből állnak. Az adattipusokat megvalósitó modulok az adatstrukturákat rejtett tipusként (hidden type) exportálják. Ez azt jelenti, hogy kivülről csak a rejtett tipus neve érhető el, de annak szerkezete, belső felépitése nem. Az adatstrukturákhoz való hozzáférés az ugyancsak exportált eljárásokon (függvény eljárásokon) keresztül történik. A definiciós modulban az eljárások fejét, az implementációs modulban pedig részletes megvalósitásukat találhatjuk. Az alábbiakban a teljesség kedvéért közöljük a 7. fejezet specifikációjában használt absztrakt adattipusokat megvalósitó modulok definiciós és implementációs részét.

A Message és Packet tipusok megvalósitásánál különleges módon jártunk el, amelynek indokai a következők. A verifikációs rendszer müködésekor a protokoll rendszer globális állapotának az entitások állapotváltozóinak VSTORAGE memóriába történő elmentése és onnan történő visszamentése igen gyakori művelet (9.2 fejezet). Célszerü tehát minimalizálni az elmentéskor és visszamentéskor mozgatott adatok mennyiségét. Az üzenetek és csomagok meghatározott formátumu adatszerkezetek, a MODULA-2 rekord fogalmával jól jel-

lemezhetők. A megvalósitásban ezen adatszerkezetek dinamikus létrehozására van szükség, amelyet a heap memóriában végzünk. A MODULA-2 rendszer azonban nem tartalmaz hulladék gyüjtőt (garbage collector) és igy a heap memória feltöredezését a felhasználónak kell megakadályozni. Ezen okból csökkenteni próbáljuk a dinamikus adatszerkezetek létrehozásának illetve megszüntetésének számát ugy, hogy üzenet és csomag könyvtárakat hozunk létre. Ezekben a könyvtárakban a különböző csomagok- illetve üzenetekből egy-egy példányt helyezünk el. Ha a protokoll specifikációjában például egy uj csomagot állitanak össze, akkor a heap memóriában ennek az összeállitásnak csak akkor lesz lenyomata (csak akkor jün létre valójában egy uj csomagot reprezentáló adatstruktura), ha az a csomag még nem szerepel a csomag könyvtárban (packetlibrary). Ez esetben az uj csomag létrehozásával egyidőben megtörténik a csomag könyvtárba sorolása is. A statikus, a verifikálás teljes ideje alatt meglévő könyvtárak bevezetése azzal az előnnyel jár, hogy a bonyolult, terjedelmes adatstrukturák egyetlen pointerrel azonosithatók, igy a globális állapot mentés és visszamentés jelentősen meggyorsul. A verifikálás félbeszakitásakor természetesen a statikus könyvtárakat is el kell menteni. A megoldás előnyei mellett nem szabad megfeledkeznünk az alapvető korlátairól sem. Problémát jelenthet a statikus könyvtárak nagy mérete. Elméletileg ugyanis nagyszámu különböző adatstruktura (pl. csomag) keletkezhet egy összetettebb strukturából. (Például egy k komponensből álló adatstrukturának $n_1 n_2 \ldots n_k$ különböző példánya létezhet, ha n_i az i-edik komponens által felvehető különböző értékek száma.) A gyakorlati tapasztalatok alapján azonban a potenciálisan lehetséges esetek közül általában csak egy részhalmaz kerül tényleges felhasználásra a protokollok müködése során. Ha ez a feltételezés nem teljesül, akkor a modulok megváltoztatásával a valódi, dinamikus adatkezelés visszaállitásával a probléma megoldható.

Az ABMessage modul a Bitalternáló protokoll üzenet típusának létrehozásáért felel. Mint azt már korábban megemlitettük (6.1 fejezet) az üzeneteket ezen a szinten (a transzparens üzenetátvitel miatt) nem kellene definiálnunk. Hogy mégis ezt tettük az,az üzenetek képernyőn történő megjelenithetőségének igényéből fakad. Nem szabad azonban elfelejtenünk, hogy ezen a szinten az üzenetek megkülönböztethetetlenek és igy verifikációs célra a környezeti modellben csak azonos üzeneteket használhatunk fel, mint ahogy az a 7.2 fejezet specifikációjából ki is tünik.

```
(*************************************************************
*                                                           *
*                  Abstract Data Type Library               *
*                                                           *
*              Alternating Bit Protocol Message             *
*                                                           *
*************************************************************)

DEFINITION MODULE ABMessage;

EXPORT QUALIFIED Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;

TYPE Message;

PROCEDURE MakeMessage(t:INTEGER):Message;

PROCEDURE MessageText(m:Message):INTEGER;

PROCEDURE EqualMessage(m1,m2:Message):BOOLEAN;

PROCEDURE NullMessage():Message;

END ABMessage.


IMPLEMENTATION MODULE ABMessage;

FROM Storage IMPORT ALLOCATE, DEALLOCATE, SetMode;
FROM QueueModule IMPORT Queue, NewQueue, Add, Middle, Length;

TYPE Message = POINTER TO INTEGER;

VAR messagelibrary:Queue;

PROCEDURE MakeMessage(t:INTEGER):Message;
  VAR m:Message;
BEGIN
```

```
  IF NOT InLibrary(t,m) THEN
    NEW(m); m^:=t; messagelibrary:=Add(messagelibrary,m);
  END;
  RETURN m;
END MakeMessage;

PROCEDURE MessageText(m:Message):INTEGER;
BEGIN RETURN m^
END MessageText;

PROCEDURE EqualMessage(m1,m2:Message):BOOLEAN;
BEGIN RETURN MessageText(m1)=MessageText(m2)
END EqualMessage;

PROCEDURE NullMessage():Message;
BEGIN RETURN MakeMessage(0)
END NullMessage;

PROCEDURE InLibrary(t:INTEGER; VAR m:Message):BOOLEAN;
VAR i:INTEGER;
BEGIN
  i:=Length(messagelibrary);
  WHILE i>0 DO
    IF MessageText(Middle(messagelibrary,i))=t
    THEN m:=Middle(messagelibrary,i); RETURN TRUE
    ELSE DEC(i)
    END;
  END;
  RETURN FALSE
END InLibrary;

BEGIN
  messagelibrary:=NewQueue();
END ABMessage.
```

Az ABPacket modul a 6.1 fejezetben tárgyalt Packet adattipus konkrét megvalósitása.

```
(*************************************************************
*                                                           *
*                  Abstract Data Type Library               *
*                                                           *
*              Alternating Bit Protocol Packet              *
*                                                           *
*************************************************************)

DEFINITION MODULE ABPacket;

FROM ABMessage IMPORT Message;
EXPORT QUALIFIED Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
```

```
TYPE Packet;

PROCEDURE MakePacket(m:Message; s:BOOLEAN):Packet;

PROCEDURE Text(pk:Packet):Message;

PROCEDURE SeqNumber(pk:Packet):BOOLEAN;

PROCEDURE EqualPacket(p1,p2:Packet):BOOLEAN;

PROCEDURE NullPacket():Packet;

END ABPacket.

IMPLEMENTATION MODULE ABPacket;

FROM SYSTEM IMPORT WORD, ADDRESS;
FROM ABMessage IMPORT Message, EqualMessage, NullMessage;
FROM Storage IMPORT ALLOCATE, DEALLOCATE, SetMode;
FROM QueueModule IMPORT Queue, NewQueue, Add, Middle,
  Length;

TYPE Packet = POINTER TO P;
     P      = RECORD
                letter:Message;
                seq   :BOOLEAN;
              END;

VAR packetlibrary:Queue;

PROCEDURE MakePacket(m:Message; s:BOOLEAN):Packet;
  VAR p:Packet;
BEGIN
  IF NOT InLibrary(m,s,p) THEN
    NEW(p);
    WITH p^ DO
      letter:=m; seq:=s;
      packetlibrary:=Add(packetlibrary,p)
    END;
  END;
  RETURN p;
END MakePacket;

PROCEDURE InLibrary( m:Message; s:BOOLEAN;
                     VAR p:Packet) : BOOLEAN;
  VAR i:INTEGER;
      mes:Message;
BEGIN
  i:=Length(packetlibrary);
  WHILE i>0 DO
    IF (ADDRESS(Text(Middle(packetlibrary,i)))=ADDRESS(m)) AND
       (SeqNumber(Middle(packetlibrary,i))=s)
    THEN p:=Middle(packetlibrary,i);
      RETURN TRUE;
    ELSE DEC(i)
    END;
```

```
    END;
    RETURN FALSE;
  END InLibrary;

  PROCEDURE Text(pk:Packet):Message;
  BEGIN RETURN pk^.letter
  END Text;

  PROCEDURE SeqNumber(pk:Packet):BOOLEAN;
  BEGIN RETURN pk^.seq
  END SeqNumber;

  PROCEDURE EqualPacket(p1,p2:Packet):BOOLEAN;
  BEGIN RETURN (ADDRESS(p1)=ADDRESS(p2))
  END EqualPacket;

  PROCEDURE NullPacket():Packet;
  BEGIN RETURN MakePacket(NullMessage(),FALSE)
  END NullPacket;


  BEGIN
    packetlibrary:=NewQueue();
  END ABPacket.
```

A QueueOfMessage, QueueOfPacket és a HistoricalQueue-OfMessage adattipusok közös megvalósitását teszi lehetővé az univerzális QueueModule modul. A SYSTEM modulból importált ADDRESS tipusnak ugyanis az a különleges sajátsága van, hogy az ADDRESS tipusu változó helyettesit minden pointer tipusu rejtett vagy expliciten kifejtett változót. A Queue-Module modul igy alkalmassá válik tetszőleges tipusu elemekből álló sor kezelésére, hiszen csak az elemekre mutató pointerek sorba állitását végzi.

```
(*************************************************************
*                                                           *
*                 Abstract Data Type Library                *
*                                                           *
*                   Universal Queue Module                  *
*                                                           *
*************************************************************)

DEFINITION MODULE QueueModule;

FROM SYSTEM IMPORT WORD, ADDRESS;
```

```
EXPORT QUALIFIED Queue, NewQueue, Add, Remove, Append,
  Clear, Front, Middle, Back, In, Empty, Equal, Length;

TYPE Queue;

PROCEDURE NewQueue() : Queue;

PROCEDURE Add(VAR q:Queue;i:ADDRESS) : Queue;

PROCEDURE Remove(VAR q:Queue) : Queue;

PROCEDURE Append(VAR q1,q2:Queue) : Queue;

PROCEDURE Clear(VAR q:Queue) : Queue;

PROCEDURE Front(q:Queue) : ADDRESS;

PROCEDURE Middle(q:Queue; seq:INTEGER) : ADDRESS;

PROCEDURE Back(q:Queue) : ADDRESS;

PROCEDURE In(q:Queue;i:ADDRESS) : BOOLEAN;

PROCEDURE Empty(q:Queue) : BOOLEAN;

PROCEDURE Equal(q1,q2:Queue;slice:INTEGER):BOOLEAN;

PROCEDURE Length(q:Queue) : INTEGER;

END QueueModule.
```

```
IMPLEMENTATION MODULE QueueModule;

FROM SYSTEM IMPORT WORD, ADDRESS;
FROM Storage IMPORT ALLOCATE, DEALLOCATE, SetMode;
FROM VerificationSystem IMPORT Error;

TYPE Queue = POINTER TO QHead;
     QP    = POINTER TO QPart;
     QHead = RECORD
               firstelem:QP;
               length : INTEGER;
             END;
     QPart = RECORD
               nextelem : QP;
               info : ADDRESS
             END;
```

```
PROCEDURE NewQueue() : Queue;
  VAR q : Queue;
BEGIN
  NEW(q);
  WITH q^ DO
    firstelem := NIL; length:=0;
  END;
  RETURN q
END NewQueue;

PROCEDURE Add(VAR q:Queue; i:ADDRESS) : Queue;
  VAR last :QP;
BEGIN
  WITH q^ DO
    INC(length);
    IF firstelem = NIL THEN
      NEW(firstelem); last:=firstelem
    ELSE last:=firstelem;
      WHILE last^.nextelem # NIL DO
        last:=last^.nextelem
      END;
      NEW(last^.nextelem);
      last:=last^.nextelem;
    END;
    last^.nextelem:=NIL; last^.info:=i
  END;
  RETURN q
END Add;

PROCEDURE Remove(VAR q:Queue) : Queue;
  VAR first : QP;
BEGIN
  WITH q^ DO
    IF firstelem # NIL THEN
      DEC(length); first:=firstelem;
      firstelem:=first^.nextelem;
      DISPOSE(first);
    ELSE Error(1); (* Remove from empty queue*)
    END;
  END;
  RETURN q
END Remove;

PROCEDURE Append(VAR q1,q2:Queue) : Queue;
  VAR index:INTEGER; act:QP;

BEGIN
  index:=1;
  WHILE index <= Length(q2) DO
    q1 := Add(q1,Middle(q2,index)); INC(index);
  END;
  RETURN q1;
END Append;
```

```
PROCEDURE Clear(VAR q:Queue) : Queue;
BEGIN
  WHILE NOT Empty(q) DO q:=Remove(q) END;
  RETURN q
END Clear;

PROCEDURE Front(q:Queue) : ADDRESS;
BEGIN
  WITH q^ DO
    IF firstelem=NIL THEN
      Error(2); (* Read from empty queue *);
      RETURN NIL
    ELSE RETURN firstelem^.info
    END
  END
END Front;

PROCEDURE Middle(q:Queue; seq:INTEGER) : ADDRESS;
  VAR i:INTEGER; act:QP;
BEGIN
  WITH q^ DO
    IF (length(seq) OR (seq(=0) THEN Error(3)
    ELSE
      act:=firstelem; i:=1;
      WHILE i(seq DO
        act:=act^.nextelem; INC(i)
      END
    END;
    RETURN act^.info;
  END
END Middle;

PROCEDURE Back(q:Queue) : ADDRESS;
 VAR last:QP;
BEGIN
  WITH q^ DO
    IF firstelem=NIL THEN
      Error(2); (* Read from empty queue *)
      RETURN NIL
    ELSE
      last:=firstelem;
      WHILE last^.nextelem # NIL DO
        last:=last^.nextelem
      END;
      RETURN last^.info
    END
  END
END Back;

PROCEDURE In(q:Queue;i:ADDRESS) : BOOLEAN;
  VAR this:QP;
BEGIN
  WITH q^ DO
    this:=firstelem;
```

```
    WHILE this # NIL DO
      IF this^.info=i THEN RETURN TRUE END;
      this:=this^.nextelem
    END
  END;
  RETURN FALSE
END In;

PROCEDURE Empty(q:Queue) : BOOLEAN;
BEGIN RETURN q^.firstelem=NIL
END Empty;

PROCEDURE Equal(q1,q2:Queue;slice:INTEGER):BOOLEAN;
  VAR i:INTEGER;
BEGIN
  IF (slice>Length(q1)) OR (slice>Length(q2)) THEN
    RETURN FALSE
  ELSE
    i:=1;
    WHILE i<=slice  DO
      IF Middle(q1,i)<>Middle(q2,i) THEN
        RETURN FALSE
      END;
      INC(i);
    END;
    RETURN TRUE;
  END;
END Equal;

PROCEDURE Length(q:Queue) : INTEGER;
BEGIN RETURN q^.length
END Length;

END QueueModule.
```

A SysTypeLibrary modul (9.2 ábra) az egyes adattipusok VSTORAGE memóriába történő elmentését és visszamentését végző eljárásokat tartalmazza. Az adattipus könyvtár bővitésekor a megfelelő modulok megfogalmazásával párhuzamosan a SysTypeLibrary modulba fel kell venni az uj adattipusok elmentéséért és visszamentéséért felelős eljárásokat is. A későbbiekben ezt a metódust egy teljesen automatikus adattipus könyvtárazó fogja felváltani.

9.1.4 A protokoll specifikáció MODULA-2 nyelvü reprezentánsa

A 7.2 fejezetben közölt protokoll példája alapján bemutatjuk a protokoll specifikáció (Függelékben elhelyezett) MODULA-2 nyelvü reprezentánsának felépitését.

A protokoll specifikációs nyelv entitás fogalmának a MODULA-2 kötött formáju modul koncepcióját feleltetjük meg. Az entitást megvalósitó modulban az entitások állapot- és interfészváltozói a modul deklarált változói. Az állapotátmeneteknek (actions) egy-egy, ezeken a változókon értelmezett eljárást feleltetünk meg, amelyben a végrehajtandó müveletek MODULA-2 nyelvü kifejtését adjuk meg. Az állapotátmenetek előfeltételeit az ActivePreCond nevü eljárásba süritjük, amelynek hivásával az adott pillanatban végrehajtható átmenetek sorszáma lekérdezhető. A RaiseAction eljárás az aktuális paramétereként megadott sorszámu állapotátmenetnek megfelelő eljárás hivásával egy adott állapotátmenet végrehajtását idézi elő. A modul SaveSystem és UnSaveSystem eljárásai az entitás aktuális állapotának a VSTORAGE memóriába (a háttértárba) történő elmentését és visszamentését végzik. A modul az állapotátmeneteknek megfelelő eljárásokon kivüli eljárásokat, valamint a deklarált változókat exportálja a rendszernek (system) megfelelő globális modul felé.

Az entitások moduljait a rendszernek (system) megfelelő külső modul fogja egységbe, amely a verifikációs rendszer felé az előzőekhez hasonló nevü és funkcióju eljárásokat exportál.

```
(*************************************************************
*                                                           *
*              MODULA-2 Representation of the               *
*                                                           *
*          Alternating Bit Protocol Specification           *
*                                                           *
*************************************************************)
```

```
DEFINITION MODULE AB;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM VirtualStorage IMPORT VSTORAGE;

EXPORT QUALIFIED RaiseActionWell, ActivePreCond,
  Wait, SaveSystem, UnSaveSystem;

PROCEDURE Wait():BOOLEAN;

PROCEDURE ActivePreCond(VAR t:TransitionSet);

PROCEDURE RaiseActionWell(entity,transition:CARDINAL):BOOLEAN;

PROCEDURE SaveSystem(s:VSTORAGE);

PROCEDURE UnSaveSystem;

END AB.
```

Az entitások közötti kapcsolatokat az állapotátmenetek végrehajtása utáni (az interfészváltozókra vonatkozó) értékadásokkal szimuláljuk. Erre szolgál a külső modul Global nevü eljárása. Az Assertion globális eljárás a protokoll rendszer elvárt tulajdonságait formalizáló boolean értékü függvényeket hordozza.

A Bitalternáló protokoll specifikáció Függelékben közölt MODULA-2 reprezentációja a 7.2 fejezet specifikációjához képest pótlólagos elemeket is tartalmaz. Az eredeti specifikáció elemzésekor ugyanis kiderült, hogy hiba esetén az ujraadások számát a protokoll nem korlátozza. A Sender entitás EnableTimeOut állapotváltozójának bevezetésével az ismétlések számát egyre korlátoztuk.

9.2 A verifikáló rendszer müködése, tapasztalatok

A futtatható, összeszerkesztett verifikáló rendszer betöltésekor, a rendszer inicializálás során a protokoll specifikációnak megfelelő modulok végrehajtható részében beállitódik az entitások állapot- és megfelelő interfészváltozói kezdeti értéke. A rendszer ezután elmenti a protokoll állapotát a VSTORAGE memóriába. Ez az állapot lesz az elérhetőségi fa gyökere. A következőkben a rendszer megvizsgálja, hogy van-e végrehajtható állapotátmenet. Ha ilyet talál, akkor közülük egyet kiválasztva végrehajtja az átmenetet és elkezdi a létrejött uj protokoll állapot elemzését, annak elmentése után. Deadlock állapotnak tekinti az uj állapotot, ha a rendszernek nincs végrehajtható átmenete (az uj állapotban) és a Wait függvény hamis értéket ad. Ha ebben a helyzetben a Wait igaz értékü, akkor az uj állapot a "wait" kvalifikációt kapja. "Invarianterror" tipusu lesz az uj állapot, ha a rendszer invariáns sértést talál. A protokoll ciklusok felderitése érdekében azután a rendszer a korábban elmentett állapotokkal hasonlitja össze az uj állapotot. Protokoll ciklus esetén az uj állapot elmentésekor feljegyződik a ciklus kezdő állapot azonositója is. Ha a rendszer az uj állapot elemzésekor nem talált deadlock vagy wait helyzeteket, ciklust vagy invariáns sértést, akkor az uj állapot a "good" minősitést kapja és a korábban már emlitett módon (a végrehajtható állapotátmenetek lekérdezésével) folytatódik a müködés. Ha az uj állapot minősitése nem "good", akkor az az elérhetőségi fa adott levelének végét jelenti. Ilyenkor a rendszer visszalép, visszamenti a protokoll rendszer korábban lementett állapotát és megvizsgálja, hogy van-e még az elérhetőségi fának az adott állapotból kiinduló be nem járt részfája. Ha van akkor a részfa felépitése kezdődik el, ha viszont nincs akkor a müködés visszalépéssel folytatódik. A verifikációs eljárás a teljes elérhetőségi fa felépitésekor fejeződik be.

A felhasználó a képernyő előtt ülve figyelemmel követheti a rendszer müködését, ahol megjelennek a végrehajtott állapotátmenetek nevei és számos egyéb információ. A felhasználó feladata az elérhetőségi fa végtelen ágainak kiszürése, abban az esetben, ha az ügyesen megválasztott invariánsok formájában megfogalmazott protokoll tulajdonságok ezt nem jeleznék (invariáns sértéssel). Lehetőség van az elérhetőségi fa a felhasználó számára érdektelen részeinek (részfáinak) kihagyására is menet közbeni parancs kiadásával. Ugyancsak felhasználói parancsra az elérhetőségi fa épités félbeszakad, ekkor a protokoll specifikáció szövege illetve a különféle változók értékei megtekinthetők.

A protokoll verifikáló rendszer müködésével kapcsolatos első tapasztalatainkat a 7.1 fejezet protokolljának elemzése során nyertük. A rendszer alapvetően jól müködik, alkalmas a protokollok korábban tárgyalt tulajdonságainak felderitésére. Kimutattuk például a 7.2 fejezetben specifikált protokoll deadlock mentességét, felderitettük ugyanakkor, hogy a protokoll 9.1.4 fejezetben emlitett módositása deadlockhoz vezet. A protokoll ugyanis nem tudja helyreállitani az egynél több csomagvesztésből származó hibát (tehát az elküldött csomag vagy a nyugtájának illetve az ujraküldött csomag vagy a nyugtájának együttes elvesztését). Első tapasztalataink szerint a rendszer müködése a vártnál lassabb, aminek alapvető oka az input-output nem elég hatékony megszervezése. Ezen és a felhasználóval történő jelenleg kissé merev kapcsolattartáson mindenképpen változtatni kell. A rendszer lassu müködésének másik nem elhanyagolható oka a mágneslemezhez fordulások nagy száma. Ezen kétféle módon kivánunk változtatni. Egyrészt minimálisra szeretnénk csökkenteni az elérhetőségi fa tárolásához szükséges memória nagyságát azzal, hogy figyelembe vesszük a 8.13 definició kapcsán elmondottakat és az összekapcsolt interfészváltozók állapotát csak egyszer tároljuk. Másrészt a VirtualStorage modul megváltoztatásával, a VSTORAGE állandóan a

memóriában tartózkodó lapja méretének növelésével (a lapméret jelenleg 256 szó hosszuságu) csökkenteni kivánjuk a diszk müveletek gyakoriságát.

A protokoll verifikáló rendszer jelenlegi formájában kisérleti eszköznek tekinthető, amelynek legfőbb oka, hogy nem rendelkezünk kellő tapasztalattal az ilyen tipusu rendszerek tervezése és használata terén. Ma még nem teljesen tisztázott az, hogy milyen szolgáltatásokat nyujtson a verifikáló rendszer, melyek a hatékony felhasználhatóság alapfeltételei, milyen legyen az input-output kezelés stb.. Reméljük azonban, hogy e rendszerrel nyert tapasztalatok hozzá fognak járulni ezen nyitott problémák megoldásához, a jövő fejlett szoftver támogatást nyujtó verifikáló rendszereinek megvalósitásához.

10. BEFEJEZÉS

10.1 Összefoglalás

A dolgozatban a számitógép-hálózatok kommunikációs protokolljainak formális specifikálásával és verifikálásával foglalkoztunk.

A protokollok a számitógép-hálózatok belső kommunikációját szabályzó elosztott algoritmusok. Ezen algoritmusok helyes vagy helytelen müködése alapvető módon befolyásolja a hálózat müködését, megbizhatóságát. A számitógép-hálózatok feltartóztathatatlan elterjedése miatt a fel nem fedezett, rejtett hibás protokoll müködésnek beláthatatlan következményei lehetnek.

A számitógép-hálózatok elosztott, paralell rendszer voltából következőleg a protokoll algoritmusok tervezése jelentős többlet nehézségeket tartalmaz a konvencionális szekvenciális algoritmusokhoz képest. Ma még a széles körben elfogadott elméleti letisztulás hiányában a protokollok tervezése sok esetben ad-hoc jellegü, a tervező szubjektumától, globális átlátó képességétől függő problémákkal terhes folyamat. Megfelelő elmélet alapok nélkül az ember egyre kevéssé lesz képes az egyre bonyolultabb igényeknek megfelelő egyre komplexebb rendszerek (protokoll rendszerek) megtervezésére és implementációjára. Olyan segédeszközökre van tehát szükség, amelyek alkalmazásával csökkenteni lehet a tervezés során fellépő problémák bonyolultságát, megteremtve ezzel a rendszerek kézbentartható tervezésének, megvalósitásának lehetőségét. Dolgozatunkban egy egységes protokoll specifikációs módszert dolgoztunk ki, amely egy speciálisan protokoll leirásra kialakitott specifikációs nyelven megfogalmazott protokollok tu-

lajdonságainak felderitését lehetővé tévő verifikációs rendszerre épül.

A csomagkapcsolt számitógép-hálózatok megvalósulásai, az egyre nagyobb nemzeti, nemzetközi hálózatok megjelenése elkerülhetetlenül nemzetközi szabványositási törekvést inkitott el, amely a 80-as évek elejére a hálózati architektura nemzetközi szabványtervezetben való rögzitéséhez vezetett. A hálózati architektura a 70-80-as évek technikai, technológiai szinvonalára épül, valószinü, hogy az uj technológiák megjelenésekor az architektura módositására, teljes felülvizsgálatára, esetleg elvetésére lesz szükség.

Dolgozatunkban a réteges hálózati architekturát az 1.4 fejezetben mutattuk be vázlatosan. Ez alapján a számitógép-hálózat egymásra rakódott protokoll rétegek együttese. A rétegek kommunikációs szolgáltatásokat nyujtanak a felettük lévő réteg(ek)nek ugy, hogy az alsó réteg(ek) szolgáltatásait kibővitik, mintegy többletszolgáltatást adva hozzá.

A 2. fejezet szakirodalmi áttekintésében a tématerület fontossabb közleményeit, cikkeit, könyveit foglaltuk össze. Ezek alapján a protokoll specifikációs és természetes módon a hozzájuk kapcsolódó verifikációs módszereket két fő és a köztük meghuzódó nem igazán harmadik csoportba sorolhatjuk. Legkorábban az állapotátmeneten alapuló módszerek alakultak ki. Legfontosabb képviselőik a véges automatá(ka)t, a különféle gráfokat alkalmazó módszerek. A programozási nyelvek a protokoll algoritmusok megfogalmazását is lehetővé teszik. A hibrid módszerek megpróbálják egyesiteni a fenti két fő irányzat előnyös tulajdonságait. A hibrid módszerek csoportjába tartoznak a dolgozat témáját alkotó specifikációs nyelvet alkalmazó protokoll leiró eljárások is.

A protokoll specifikáció alapvető célja az ember-em-

ber közötti információ csere megvalósitása. Ebből következőleg a leiró formalizmusnak számos követelményt kell kielégitenie. A specifikációs módszerek elvárt tulajdonságait a 3.1 fejezetben foglaltuk össze. A jelenleg használt és valószinüleg a közeljövőben elterjedő specifikációs metódusok sajnos még tág teret kinálnak a hibás ("elemi hibákkal terhelt") protokollok megfogalmazásának. Mindenképpen szükség van tehát a protokollok helyességét bizonyitó illetve ellenőrző verifikációs módszerekre. A protokoll verifikációs feladat tipusokat és a verifikáció alapproblémáját mutattuk be a 3.2 részben. A specifikációval és verifikációval szemben támasztott általános követelmények a specifikációs nyelvvel szemben támasztott elvárásokká konvertálhatók. Az ideális protokoll leiró nyelv magasszintü, moduláris felépitésü, párhuzamos, nemdeterminisztikus, erősen tipusos nyelv.

A következő fejezet a protokoll tervezés javasolt eljárását mutatja be. A tervező a specifikációs nyelven megfogalmazott protokoll statikus hibáit az elemző- forditóprogrammal mutatja ki, majd az ezen hibáktól mentes protokoll dinamikus elemzését a verifikáló rendszer segitségével végzi el. A módszer alkalmazásától azt várjuk, hogy nagymértékben egyszerüsödik a protokoll tervezés ma még eléggé komplex folyamata és a maiaknál megbizhatóbb protokoll definiciók jönnek létre.

A számitógép-hálózati kommunikációs protokollok matematikai modelljére tettünk javaslatot az 5. fejezetben. A protokollt névvel ellátott, particionált állapotátmenet rendszernek fogtuk fel. Definiáltuk az ilyen rendszerek müködésének fogalmát, állapot sorozatnak tekintve azt. A párhuzamos rendszerek leirásánál a paralell rendszerek nemdeterminisztikus szekvenciális rendszerekké transzformációját követtük. Kimutattuk azonban e megközelitési mód korlátait is. Az elmélet értékü protokoll specifikáció definició mellett a gyakorlatban használható reprezentációt is bemutattuk. Bevezettük a k-szintü állapotátmenet rendszer

fogalmát, amely a protokoll különféle absztrakciós szintü modell sorozata. A gyenge és erős protokoll ekvivalencia értelmezésével zárult a fejezet.

A protokoll elméleti modelljének egy programozási nyelvi megvalósitását tárgyaltuk a következőkben. A specifikációs nyelv két fő rétegét a MODULA-2 nyelven megfogalmazott absztrakt adattipus könyvtár és a speciális nyelvi elemek rétege alkotja. A nyelv a protokollt, protokoll rendszert entitások halmazaként definiálja. Az entitások között statikus kapcsolatok épülnek ki, melyeket közös, osztott változókkal modellezünk.

A 6.1 fejezetben az absztrakt adattipusok értelmezéséről illetve a különféle szemantika definiálási lehetőségekről beszéltünk, majd az általunk alkalmazott procedurális megközelitési módot elemeztük. Példákon keresztül illusztráltuk az elmondottakat.

Az entitás mint absztrakt objektum belső szerkezete állapot- és interfészváltozó deklarációra, névvel és előfeltétellel ellátott átmenetekre, valamint a kezdeti állapotot beállitó inicializáló részre tagolódik. A nyelv lehetőséget teremt az entitások között kialakuló kapcsolatok bizonyos konzisztenciájának ellenőrzésére is.

A protokoll rendszer algoritmikus környezetének leirása entitásokkal, interfészváltozókkal történik. A környezeti leirás a specifikált protokoll (réteg) alatt és felett elhelyezkedő protokoll rétegek modellje, a tőlük elvárt illetve feltételezett tulajdonságok koncentrált kifejezési formája. A protokolltól magától elvárt tulajdonságok megfogalmazása protokoll invariánsok felállitásával lehetséges.

A 6.5 fejezetben bemutattuk a javasolt protokoll specifikációs nyelv egy lehetséges szintaxisát. A nyelv konkrét alkalmazására egy viszonylag egyszerü, könnyen érthető de nem triviális példát közöltünk.

A 8. fejezet a protokollok irodalomban fellelhető főbb "fenomenologikus" tulajdonságait vizsgálta meg. Először szöveges értelmezéseket adtunk, majd felhasználva az 5. fejezet matematikai modelljét megalkottuk a formális definiciókat is. A protokoll tulajdonságok formalizálásánál a tulajdonságok rendszer invariánsok formájában történő kifejtését követtük, megteremtve ezzel azt, hogy egy általános invariancia érvényességi vizsgálati módszer a protokoll tulajdonságok felderitését is lehetővé tegye. Az adott elméleti kereten belül a progress tulajdonság definiálása nem vezetett eredményre. A 8.3 fejezetben mutattuk be az invariánsok érvényességi vizsgálatát végző állapotelérhetőségi analizis módszert. A következő fejezetben ezen módszert konkrétan megvalósitó interaktiv protokoll verifikáló rendszer szerkezetét, müködését és használatát tárgyaltuk. A rendszer nemcsak az alapvető protokoll tulajdonságok megállapitására alkalmas, hanem ha a tervező az adott formalizmus keretén belül képes a protokoll helyesség megfogalmazására, akkor a helyesség is ellenőrizhetővé válik. A rendszer felhasználásának kezdeti tapasztalatai rávilágitottak néhány hiányosságra, amelyeknek kijavitása gyorsabb futást, hatékonyabb, kényelmesebb használatot eredményez.

10.2 A továbblépés irányai

Az alábbiakban néhány nyitott probléma felvetésével a továbblépés lehetséges irányairól szólunk.

Az egyik legalapvetőbb probléma ugy gondoljuk, hogy továbbra is a protokollok matematikai modelljének kérdése. Olyan formalizmusra lenne szükségünk, amely lehetőséget teremt az alapvető tulajdonságok megfogalmazására, meglétük vagy hiányuk bizonyitására. A probléma szoros kapcsolatban

van az un. statikus definiciók nem csupán a számitógép-hálózatok területére kiterjedő kérdéskörével. A statikus definició, ellentétben a müvelet orientált definicióval (operational definition) a MIT kérdésre felel és nem rögziti a HOGYAN-t. A jelenleg alkalmazott definiciós technikáink tulnyomórészt müvelet orientáltak, azt a módot (utat) rögzitik, amelynek betartása az elérendő cél megvalósulását eredményezi. Például ezért nehéz formalizálni a dolgozatban tárgyalt elméleti kereten belül a progress tulajdonságot ellentétben a deadlockkal, hiszen a holtpont bizonyos müködés (állapot sorozat) kényszeritésével elkerülhető. A továbblépés egyik utjának tünik a statikus jellegü modális logikai definiciós módszerek nem régen megindult kifejlesztése.

Ugyancsak nagy lehetőségeket látunk a számitógéppel támogatott verifikációs rendszereken kivül az automatikus protokoll szintetizáló rendszerek megvalósitásában is. Ilyen rendszerek nemcsak a protokoll specifikáció, hanem az automatikus protokoll implementáció előállitásánál is szerepet játszhatnak, bár ugy gondoljuk, hogy az automatikus kódgenerálás nem a közeljövő feladata.

Mindezek alapján ugy gondoljuk, hogy az uj elméleti eredményeknek mihamarabb uj specifikációs nyelvi javaslatokban kell testet ölteniük, megkönnyitve ezzel az eredmények gyakorlati felhasználását.

IRODALOM

[Aho és m. 82] Aho V.A.-Hopcroft J.E.-Ullman J.D.:
Számitógép-algoritmusok tervezése és analizise.
Müszaki Könyvkiadó, Budapest 1982

[Alfonzetti és m. 79] Alfonzetti S.-Casale S.-Faro A.:
A formal description of the DTE packet level in the X.25 Recommendation.
Alta Frequenza, Vol. XLVIII - N.8 AGOSTO 1979

[Alfonzetti és m. 80] Alfonzetti S.-Casale S.-Faro A.:
X.25 DTE level 3 formalization using the "Second Theory of Colloquies".
Proc. IEEE ICCC 80, New York 1980

[ASZH 81] Az Akadémiai Számitógép-Hálózat referencia modellje.
Kézirat, MTA-SZTAKI Hálózat és Távadatfeldolgozási O.

[Birkhoff és m. 74] Birkhoff G.-Bartee T.C.:
A modern algebra a számitógép-tudományban.
Müszaki Könyvkiadó, Budapest 1974

[Bochmann 75] Bochmann G.V.:
Logical verification and implementation of protocols.
Proc. Fourth Data Comm. Symp. ACM/IEEE, 1975

[Bochmann és m. 77a] Bochmann G.V.-Gecsei J.:
A unified method for the specification and verification of protocols.
Proc. Information Processing 77 IFIP, Toronto 1977

[Bochmann és m. 77b] Bochmann G.V-Chung R.J.:
A formalized specification of HDLC classes of procedures.
Proc. of the Nat. Telecomm. Conf. Los Angeles 1977

[Bochmann 78] Bochmann G.V.:
Finite state description of communication protocols.
Proc. Comp. Netw. Prot. Liege 1978

[Bochmann és m. 79] Bochmann G.V.-Joachim T.:
Development and structure of an X.25 implementation.
IEEE Trans. on Soft. Eng. Vol. SE-5, No. 5 Sep. 1979

[Bochmann 80] Bochmann G.V.:
A general transition model for protocols and communication services.
IEEE Trans. on Comm. Vol. COM-28, No. 4, Apr. 1980

[Bochmann és m. 80] Bochmann G.V.-Sunshine C.A.:
Formal methods in communication protocol design.
IEEE Trans. on Comm. Vol. COM-28, No. 4. Apr. 1980

[Böszörményi 79] Böszörményi L.:
Néhány müködő X.25 implementáció irodalmának feldolgozása.
VEIKI 50.99-001, Tanulmány 1979

[Böszörményi 81] Böszörményi L.:
Multi-task rendszerek fejlesztése magasszintü nyelven.
MTA-SZTAKI Tanulmány 128/1981

[Brand és m. 78] Brand D.-Joyner W.H.:
Verification of protocols using symbolic execution.
Proc. Comp. Netw. Prot. Liege 1978

[Brinch Hansen 75] Per Brinch Hansen:
Concurrent pascal report.
Inf. Sci. Calif. Inst. of Techn. 1975

[Dahl és m. 78] Dahl O.J-Dijkstra E.W.-Hoare C.A.R.:
Strukturált programozás.
Müszaki Könyvkiadó, Budapest, 1978

[Danthine 78] Danthine A.A.S.:
Protocol representation with finite-state models.
IEEE Trans. on Comm. Vol. COM-28, No. 4. Apr. 1978

[Danthine és m. 78] Danthine A.A.S.-Bremer J.:
Modelling and verification of end-to-end transport protocols.
Proc. Comp. Netw. Prot. Liege 1978

[Davies és m. 78] Davies D.W.-Barber D.L.A.:
Számitógép-hálózatok.
Müszaki Könyvkiadó, Budapest, 1978

[Davies és m. 82] Davies D.W.-Barber D.L.A.-Price W.L.-Solomonides C.M.:
Számitógép-hálózatok és protokollok.
Müszaki Könyvkiadó, Budapest, 1982

[ETHERNET 80] The Ethernet, a local area network Data link layer and Physical layer specifications. Version 1.0
DEC-INTEL-XEROX 1980

[Geissmann 81] Geissmann L.:
User guide to MODULA-2 system.
ETH-IFI Zürich 1981

[Guttag 80] Guttag J.:
Notes on type abstraction Version 2.
IEEE Trans. on Soft. Eng. Vol. SE-6, No. 1. Jan. 1980

[Hailpern és m. 80] Hailpern B.-Owicki S.:
Verifying network protocols using temporal logic.
NBS Trends and Appl. Symp. 1980

[Harangozó 78a] Harangozó J.:
Protocol definition with formal grammars.
Proc. Comp. Netw. Prot. Liege 1978

[Harangozó 78b] Harangozó J.:
Számitógép-hálózatok protokolljainak formális leirása.
Kandidátusi értekezés, Budapest, 1978

[Hoare 78] Hoare C.A.R.:
Communicating sequential processes.
Comm. of the ACM Vol. 21. No. 8. Aug. 1978

[ISO 81a] ISO TC 97 / SC 16:
Data processing - open systems interconnection - basic reference model.
Comp. Netw. 5 1981

[ISO 81b] ISO TC 97 / SC 16 / WG 1 Subgroup B:
A FDT based on an extended state transition model.
Working Draft, Boston, Dec. 1981

[Keller 76] Keller R.M.:
Formal verification of parallel programs.
Comm. of the ACM Vol. 19. No. 7. Jul. 1976

[Kovács 81a] Kovács L.:
Bevezetés a CONCURRENT PASCAL programozási nyelvbe.
Korszerü folyamatirányitó rendszerek tervezése,
Szakszemináriumi füzet 1, BME Folyamatszabályozási Tanszék, Budapest 1981

[Kovács 81b] Kovács L.:
A számitógép-hálózatok alapjai.
Korszerü folyamatirányitó rendszerek tervezése,
Szakszemináriumi füzet 3, BME Folyamatszabályozási Tanszék, Budapest 1981

[Kovács 82a] Kovács L.:
Protokollok formális leirása és verifikálása.
Előadás, JATE Kalmár László Kib. Lab. és a KFKI MSZKI közös szemináriuma, Szeged 1982

[Kovács 82b] Kovács L.:
Egy számitógép-hálózati protokoll formális leirása és helyességének ellenőrzése.
Előadás, Számitógép-hálózatok kommunikációs protokolljai, Tavaszi Iskola Siófok 1982

[Kovács 82c] Kovács L.:
Egy számitógép-hálózati protokoll formális leirása specifikációs nyelv segitségével és helyességének ellenőrzése állapotelérhetőségi vizsgálattal.
MTA-SZTAKI Working Paper, Budapest 1982; Inf. Elektr. (megjelenés alatt)

[Le Moli 73] Le Moli G.:
A theory of colloquies.
Alta Frequenza, Vol. XLII 1973

[Leveille és m. 82] Leveille A.-Bochmann G.V.:
Formal specification of a transport protocol.
Working Draft, 1982

[Margitics 81] Margitics I.:
Heterogén számitógép-hálózat end-to-end transzport protokolljának ismertetése és elemzése.
Tanulmány, 1981 Budapest

[Merlin 76a] Merlin Ph.M.:
A methodology for the design and implementation of communication protocols.
IEEE Trans. on Comm. Vol. Com-24, No. 6. Jun. 1976

[Merlin 76b] Merlin Ph.M.-Farber D.J.:
Recoverability of communication protocols - implications of a theoretical study.
IEEE Trans. on Comm. Vol. COM-24, Sep. 1976

[Merlin 79] Merlin Ph.M.:
Specification and validation of protocols.
IEEE Trans. on Comm. Vol. COM-27, No. 11. Nov. 1979

[Molnár 81] Molnár I.:
A strukturált rendszertervezés helyzetének elemzése.
Korszerü folyamatirányitó rendszerek tervezése,
Szakszemináriumi füzet 2, BME Folyamatszabályozási
Tanszék, Budapest 1981

[Piatkowski 79] Piatkowski T.F.:
A formal definition of the ISO-ANSI Open Systems Interconnection Reference Model - a brief overview and
proposal for a state-oriented systems approach.
Proc. The 1st Internat. Conf. on Distr. Computing
Systems, Huntsville, Alabama 1979

[Pouzin és m. 78] Pouzin L.-Zimmermann H.:
A tutorial on protocols.
Proc. of the IEEE, Vol. 66, No. 11. Nov. 1978

[Razouk és m. 80] Razouk R.R.-Estrin G.:
Modeling and verification of communication protocols
in SARA: The X.21 interface.
IEEE Trans. on Comp, Vol. C-29, No. 12. Dec. 1980

[Rowe 80] Rowe L.A.:
Data abstraction from a programming language viewpoint.
1980 ACM 0-89791-031-1/80

[Rudin és m. 78] Rudin H.-West C.H.-Zafiropulo P.:
Automated protocol validation one chain of development.
Proc. Comp. Netw. Prot. Conf. Liege 1978

[Schultz és m. 80] Schultz G.D.-Rose D.B.-West C.H.-Gray J.P.:
Executable description and validation of SNA.
IEEE Trans. on Comm. Vol. COM-28, No. 4. Apr. 1980

[Schwabe 81] Schwabe D.:
Formal techniques for the specification and verification of protocols.
Technical Report, UCLA Apr. 1981

[Schwartz és m. 81] Schwartz R.L.-Melliar-Smith P.M.:
Temporal logic specification of distributed systems.
Proc. of the Sec. Int. Conf. on Distr. Syst. INRIA Paris 1981

[Sidhu 82] Sidhu D.P.:
Rules for synthesizing correct communication protocols.
Comp. Comm. Rev. Vol. 12. No. 1. Jan. 1982

[Stenning 76] Stenning V.N.:
A data transfer protocol.
Comp. Netw. Vol. 1. Sep. 1976

[Stenning 79] Stenning V.N.:
Definition and verification of computer network protocols.
Nat. Phys. Lab. Report DNACS 15/79 Feb. 1979

[Sunshine 78] Sunshine C.A.:
Survey of protocol definition and verification techniques.
Proc. Comp. Netw. Prot. Conf. Liege 1978

[Sunshine 79] Sunshine C.A.:
Formal techniques for protocol specification and verification.
Computer, Sep. 1979

[Tavaszi 80] Számitógép-hálózat modellek, Tavaszi Iskola, Szeged 1980

[Tavaszi 82] Kommunikációs protokollok, Tavaszi Iskola, Siófok 1982

[Teng és m. 78a] Teng A.Y.-Liu M.T.:
A formal approach to the design and implementation of network communication protocols.
Proc. of the 2nd Int. Comp. Soft. and Appl. Conf. IEEE Chicago, Illinois 1978

[Teng és m. 78b] Teng A.Y.-Liu M.T.:
A formal model for automatic implementation and logical validation of network communication protocols.
Proc. of the Comp. Netw. Symp. NBS Gaithersburg, Maryland 1978

[Timothy és m. 68] Timothy L.K.-Bona B.E.:
State space analysis: an introduction.
Mc. Graw-Hill 1968

[Zafiropulo 78] Zafiropulo P.:
Protocol validation by duologue-matrix analysis.
IEEE Trans. on Comm. Vol. COM-26. No. 8. Aug 1978

[Zafiropulo 78b] Zafiropulo P.:
Design rules for producing logically complete two-process interactions and communications protocols.
Proc. 2nd Int. Conf. Comput. Software and Appl. IEEE Chicago, Illinois 1978

[Zafiropulo és m.80] Zafiropulo P.-West C.H.-Rudin H.-Cowan D.D-Brand D.:
Towards analyzing and synthesizing protcols.
IEEE Trans. on Comm. Vol. COM-28. No. 4. Apr. 1980

[van-Mierop 79] van-Mierop D.:
Design and verification of distributed interacting processes.
Technical Report, UCLA March 1979

[Varga 80a] Varga L.:
Specifications of reliable software.
Operating Systems, VI. Winter School, Visegrád 1980

[Varga 80b] Varga L.:
Rendszerprogramok elmélete és gyakorlata.
Akadémiai Kiadó, Budapest 1980

[Varga 81] Varga L.:
Programok analizise és szintézise.
Akadémiai Kiadó, Budapest 1981

[West 78] West C.H.:
An automated technique of communication protocol validation.
IEEE Trans. on Comm. Vol. COM-26. No. 8. Aug 1978

[Wirth és m. 75] Jensen K.-Wirth N.:
Pascal user manual and report.
Springer-Verlag, 1975

[Wirth 80] Wirth N.:
MODULA-2
ETH-IFI, Zürich 1980

[Wirth 82] Wirht N.:
Algoritmusok + adatstrukturák = programok.
Müszaki Könyvkiadó Budapest, 1982

FÜGGELÉK

```
(*************************************************************
*                                                           *
*              MODULA-2 Representation of the               *
*                                                           *
*          Alternating Bit Protocol Specification           *
*                                                           *
*************************************************************)

DEFINITION MODULE AB;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM VirtualStorage IMPORT VSTORAGE;

EXPORT QUALIFIED RaiseActionWell, ActivePreCond,
  Wait, SaveSystem, UnSaveSystem;

PROCEDURE Wait():BOOLEAN;

PROCEDURE ActivePreCond(VAR t:TransitionSet);

PROCEDURE RaiseActionWell(entity,transition:CARDINAL):BOOLEAN;

PROCEDURE SaveSystem(s:VSTORAGE);

PROCEDURE UnSaveSystem;

END AB.


IMPLEMENTATION MODULE AB;

IMPORT TreeStreamHandling, ABMessage, ABPacket, QueueModule,
  VirtualStorage, VInOut,MultIO, VFiles, SysTypeLibrary,
  SYSTEM;
FROM TreeStreamHandling IMPORT TransitionSet;
FROM QueueModule IMPORT Queue, NewQueue, Add, Remove,
  Append, Clear, Front, Middle, Back, In, Empty,
  Equal, Length;
FROM ABMessage IMPORT Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;
FROM ABPacket IMPORT Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
FROM SysTypeLibrary IMPORT SaveMessage, UnSaveMessage,
  SavePacket, UnSavePacket, SaveQueue, UnSaveQueue;
FROM SYSTEM IMPORT ADDRESS, WORD;
FROM VirtualStorage IMPORT VSTORAGE;

VAR m:Message;

MODULE Sender;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM QueueModule IMPORT Queue, NewQueue, Add, Remove,
  Append, Clear, Front, Middle, Back, In, Empty,
  Equal, Length;
```

```
FROM ABMessage IMPORT Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;
FROM ABPacket IMPORT Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
FROM SysTypeLibrary IMPORT SaveMessage, UnSaveMessage,
  SavePacket, UnSavePacket, SaveQueue, UnSaveQueue;
FROM SYSTEM IMPORT ADDRESS, WORD;
FROM VirtualStorage IMPORT VSTORAGE, WriteWord, ReadWord;
FROM VFiles IMPORT treeStorage;

IMPORT Global;

EXPORT QUALIFIED SSN, Pending, Sent, InPort, OutPort,
  MessageInTransit, MessageToSend, ActivePreCond,
  RaiseAction, SaveSystem, UnSaveSystem;

VAR
(* State variables *)
    SSN              : BOOLEAN;
    Pending          : Queue;    (*OfPacket *)
    Sent             : Queue;    (*OfMessage*)
    EnableTimeout    : BOOLEAN;
(* Interfaces exported *)
    MessageToSend    : Queue;        (*OfMessage*)
    MessageInTransit : BOOLEAN;
    InPort, OutPort  : Queue;        (*OfPacket *)

PROCEDURE ActivePreCond(VAR t:BITSET);
BEGIN
  IF (NOT Empty(MessageToSend)) AND
    (NOT MessageInTransit)
  THEN
    INCL(t,0)
  END;
  IF NOT Empty(InPort) THEN
    INCL(t,1)
  END;
  IF (NOT Empty(Pending) AND EnableTimeout) THEN
    INCL(t,2)
  END;
END ActivePreCond;

PROCEDURE UserSend;
BEGIN
  Pending:=Add(Pending,
    ADDRESS(MakePacket(Message(Front(MessageToSend)),SSN)));
  Sent:=Add(Sent,Front(MessageToSend));
  MessageInTransit:=TRUE;
  OutPort:=Add(OutPort,
    ADDRESS(MakePacket(Message(Front(MessageToSend)),SSN)));
  EnableTimeout:=TRUE;
  MessageToSend:=Remove(MessageToSend);
END UserSend;
```

```
PROCEDURE Receive;
BEGIN
  IF SeqNumber(Packet(Front(InPort)))=SSN THEN
    SSN:=NOT SSN;
    Pending:=Clear(Pending);
    MessageInTransit:=FALSE;
  END;
  InPort:=Remove(InPort);
  EnableTimeout:=TRUE;
END Receive;

PROCEDURE TimeOut;
BEGIN
  OutPort:=Append(OutPort,Pending);
  EnableTimeout:=FALSE;
END TimeOut;

PROCEDURE RaiseAction(transition:CARDINAL) : BOOLEAN;
BEGIN
  CASE transition OF
    0 : UserSend
   |1 : Receive
   |2 : TimeOut
  ELSE RETURN FALSE
  END;
  Global(0);
  RETURN TRUE
END RaiseAction;

PROCEDURE SaveSystem(s:VSTORAGE);
BEGIN
  WriteWord(s,SSN);
  SaveQueue(Pending,s);
  SaveQueue(Sent,s);
  SaveQueue(InPort,s);
  SaveQueue(OutPort,s);
  WriteWord(s,MessageInTransit);
  SaveQueue(MessageToSend,s);
  WriteWord(s,EnableTimeout);
END SaveSystem;

PROCEDURE UnSaveSystem;
  VAR w:WORD;
BEGIN
  ReadWord(treeStorage,w);
  SSN:=BOOLEAN(w);
  UnSaveQueue(Pending,treeStorage);
  UnSaveQueue(Sent,treeStorage);
  UnSaveQueue(InPort,treeStorage);
  UnSaveQueue(OutPort,treeStorage);
  ReadWord(treeStorage,w);
  MessageInTransit:=BOOLEAN(w);
  UnSaveQueue(MessageToSend,treeStorage);
  ReadWord(treeStorage,w);
  EnableTimeout:=BOOLEAN(w);
END UnSaveSystem;
```

```
BEGIN
  SSN:=FALSE;
  Pending:=NewQueue();
  Sent:=NewQueue();
  EnableTimeout:=TRUE;
  MessageToSend:=NewQueue();
  MessageInTransit:=FALSE;
  InPort:=NewQueue();
  OutPort:=NewQueue();
END Sender;



MODULE Receiver;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM QueueModule IMPORT Queue, NewQueue, Add, Remove,
  Append, Clear, Front, Middle, Back, In, Empty,
  Equal, Length;
FROM ABMessage IMPORT Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;
FROM ABPacket IMPORT Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
FROM SysTypeLibrary IMPORT SaveMessage, UnSaveMessage,
  SavePacket, UnSavePacket, SaveQueue, UnSaveQueue;
FROM SYSTEM IMPORT ADDRESS, WORD;
FROM VirtualStorage IMPORT VSTORAGE, WriteWord, ReadWord;
FROM VFiles IMPORT treeStorage;
IMPORT Global;

EXPORT QUALIFIED RSN, ReceiveBuffer, Received, InPort,
  OutPort, MessageReceived, UserWishesToReceive,
  SaveSystem, UnSaveSystem, ActivePreCond, RaiseAction;

VAR
(* State variables *)
    RSN           : BOOLEAN;
    ReceiveBuffer: Queue;     (*OfPacket *)
    Received      : Queue;     (*OfMessage*)
(* Interfaces exported *)
    MessageReceived      : Queue;     (*OfMessage*)
    UserWishesToReceive : BOOLEAN;
    InPort,OutPort       : Queue;     (*OfPacket *)

PROCEDURE ActivePreCond(VAR t:BITSET);
BEGIN
  IF (NOT Empty(MessageReceived)) AND
    UserWishesToReceive
  THEN
    INCL(t,0)
  END;
  IF NOT Empty(InPort) THEN
    INCL(t,1)
  END;
END ActivePreCond;
```

```
PROCEDURE UserRcv;
BEGIN
  Received:=Add(Received,
    ADDRESS(Text(Packet(Front(ReceiveBuffer)))));
  MessageReceived:=Remove(MessageReceived);
(*UserWishesToReceive:=FALSE;*)
  OutPort:=Append(OutPort,ReceiveBuffer);
  ReceiveBuffer:=Remove(ReceiveBuffer);
END UserRcv;

PROCEDURE Receive;
BEGIN
  IF SeqNumber(Packet(Front(InPort)))=RSN THEN
    ReceiveBuffer:=Add(ReceiveBuffer,ADDRESS(Front(InPort)));
    MessageReceived:=Add(MessageReceived,ADDRESS(Front(InPort)));
  END;
  IF (SeqNumber(Packet(Front(InPort)))<>RSN) AND
    Empty(ReceiveBuffer)
  THEN
    OutPort:=Add(OutPort,Front(InPort));
  END;
  IF SeqNumber(Packet(Front(InPort)))=RSN THEN RSN:=NOT RSN END;
  InPort:=Remove(InPort);
END Receive;

PROCEDURE SaveSystem(s:VSTORAGE);
BEGIN
  WriteWord(s,RSN);
  SaveQueue(ReceiveBuffer,s);
  SaveQueue(Received,s);
  SaveQueue(InPort,s);
  SaveQueue(OutPort,s);
  WriteWord(s,UserWishesToReceive);
  SaveQueue(MessageReceived,s);
END SaveSystem;

PROCEDURE UnSaveSystem;
  VAR w:WORD;
BEGIN
  ReadWord(treeStorage,w);
  RSN:=BOOLEAN(w);
  UnSaveQueue(ReceiveBuffer,treeStorage);
  UnSaveQueue(Received,treeStorage);
  UnSaveQueue(InPort,treeStorage);
  UnSaveQueue(OutPort,treeStorage);
  ReadWord(treeStorage,w);
  UserWishesToReceive:=BOOLEAN(w);
  UnSaveQueue(MessageReceived,treeStorage);
END UnSaveSystem;


PROCEDURE RaiseAction(transition:CARDINAL):BOOLEAN;
BEGIN
  CASE transition OF
    0 : UserRcv
```

```
    :1 : Receive
  ELSE RETURN FALSE
  END;
  Global(1);
  RETURN TRUE
END RaiseAction;

BEGIN
  RSN:=FALSE;
  ReceiveBuffer:=NewQueue();
  Received:=NewQueue();
  MessageReceived:=NewQueue();
  InPort:=NewQueue();
  OutPort:=NewQueue();
END Receiver;



MODULE Data;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM QueueModule IMPORT Queue, NewQueue, Add, Remove,
  Append, Clear, Front, Middle, Back, In, Empty,
  Equal, Length;
FROM ABMessage IMPORT Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;
FROM ABPacket IMPORT Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
FROM SysTypeLibrary IMPORT SaveMessage, UnSaveMessage,
  SavePacket, UnSavePacket, SaveQueue, UnSaveQueue;
FROM SYSTEM IMPORT ADDRESS, WORD;
FROM VirtualStorage IMPORT VSTORAGE, WriteWord, ReadWord;
FROM VFiles IMPORT treeStorage;
IMPORT Global;

EXPORT QUALIFIED Buffer, SaveSystem, UnSaveSystem,
  ActivePreCond, RaiseAction;

VAR
(* Interfaces exported *)
  Buffer:Queue;     (*OfPacket*)

PROCEDURE ActivePreCond(VAR t:BITSET);
BEGIN
  IF NOT Empty(Buffer) THEN INCL(t,0) END;
END ActivePreCond;

PROCEDURE RaiseAction(transition:CARDINAL) : BOOLEAN;
BEGIN
  CASE transition OF
    0 : LosePacket
  ELSE RETURN FALSE
  END;
  Global(2);
  RETURN TRUE
END RaiseAction;
```

```
PROCEDURE LosePacket;
BEGIN
  Buffer:=Remove(Buffer);
END LosePacket;

PROCEDURE SaveSystem(s:VSTORAGE);
BEGIN
  SaveQueue(Buffer,s);
END SaveSystem;

PROCEDURE UnSaveSystem;
BEGIN
  UnSaveQueue(Buffer,treeStorage);
END UnSaveSystem;

BEGIN
  Buffer:=NewQueue();
END Data;



MODULE Ack;

FROM TreeStreamHandling IMPORT TransitionSet;
FROM QueueModule IMPORT Queue, NewQueue, Add, Remove,
  Append, Clear, Front, Middle, Back, In, Empty,
  Equal, Length;
FROM ABMessage IMPORT Message, MakeMessage, MessageText,
  EqualMessage, NullMessage;
FROM ABPacket IMPORT Packet, MakePacket, Text, SeqNumber,
  EqualPacket, NullPacket;
FROM SysTypeLibrary IMPORT SaveMessage, UnSaveMessage,
  SavePacket, UnSavePacket, SaveQueue, UnSaveQueue;
FROM SYSTEM IMPORT ADDRESS, WORD;
FROM VirtualStorage IMPORT VSTORAGE, WriteWord, ReadWord;
FROM VFiles IMPORT treeStorage;
IMPORT Global;

EXPORT QUALIFIED Buffer, SaveSystem, UnSaveSystem,
  ActivePreCond, RaiseAction;

VAR
(* Interfaces exported *)
  Buffer:Queue;            (*OfPacket*)

PROCEDURE ActivePreCond(VAR t:BITSET);
BEGIN
  IF NOT Empty(Buffer) THEN INCL(t,0) END;
END ActivePreCond;

PROCEDURE RaiseAction(transition:CARDINAL) : BOOLEAN;
BEGIN
  CASE transition OF
    0 : LosePacket
  ELSE RETURN FALSE
  END;
```

```
  Global(2);
  RETURN TRUE
END RaiseAction;

PROCEDURE LosePacket;
BEGIN
  Buffer:=Remove(Buffer);
END LosePacket;

PROCEDURE SaveSystem(s:VSTORAGE);

BEGIN
  SaveQueue(Buffer,s);
END SaveSystem;

PROCEDURE UnSaveSystem;
BEGIN
  UnSaveQueue(Buffer,treeStorage);
END UnSaveSystem;

BEGIN
  Buffer:=NewQueue();
END Ack;


VAR
(* Interfaces exported *)
  MessageToABProtocol, MessageFromABProtocol:Queue;
  NotOkToSend, OkToReceve:BOOLEAN;


PROCEDURE Global(entity:INTEGER);
BEGIN
  CASE entity OF
    0 : Data.Buffer           :=Sender.OutPort;
        Receiver.InPort       :=Sender.OutPort;
        Ack.Buffer            :=Sender.InPort;
        Receiver.OutPort      :=Sender.InPort;

    1 : Ack.Buffer            :=Receiver.OutPort;
        Sender.InPort         :=Receiver.OutPort;
        Data.Buffer           :=Receiver.InPort;
        Sender.OutPort        :=Receiver.InPort;

    2 : Sender.OutPort        :=Data.Buffer;
        Receiver.InPort       :=Data.Buffer;

    3 : Sender.InPort         :=Ack.Buffer;
        Receiver.OutPort      :=Ack.Buffer;
  ELSE
  END;
END Global;
```

```
PROCEDURE ActivePreCond(VAR t:TransitionSet);
BEGIN
  Sender.ActivePreCond(t[0]);
  Receiver.ActivePreCond(t[1]);
  Data.ActivePreCond(t[2]);
  Ack.ActivePreCond(t[3]);
END ActivePreCond;

PROCEDURE RaiseActionWell(entity,transition:CARDINAL):BOOLEAN;
  VAR ok:BOOLEAN;
BEGIN
  CASE entity OF
    0 : ok:= Sender.RaiseAction(transition);
   |1 : ok:= Receiver.RaiseAction(transition);
   |2 : ok:= Data.RaiseAction(transition);
   |3 : ok:= Ack.RaiseAction(transition);
  ELSE RETURN FALSE
  END;
  RETURN (ok AND Assertion());
END RaiseActionWell;

PROCEDURE SaveSystem(s:VSTORAGE);
BEGIN
  Sender.SaveSystem(s);
  Receiver.SaveSystem(s);
  Data.SaveSystem(s);
  Ack.SaveSystem(s);
END SaveSystem;

PROCEDURE UnSaveSystem;
BEGIN
  Sender.UnSaveSystem;
  Receiver.UnSaveSystem;
  Data.UnSaveSystem;
  Ack.UnSaveSystem;
END UnSaveSystem;

PROCEDURE Assertion():BOOLEAN;
BEGIN
  IF (Sender.SSN()Receiver.RSN) AND
    Empty(Receiver.MessageReceived) OR
    (Sender.SSN=Receiver.RSN) AND
    Empty(Sender.Pending) THEN RETURN
    (Equal(Sender.Sent,Receiver.Received,Length(Sender.Sent))
    AND TRUE)
  ELSE RETURN
    (Equal(Sender.Sent,Receiver.Received,Length(Sender.Sent)-1)
    AND EqualMessage(Message(Back(Sender.Sent)),
    Text(Packet(Front(Sender.Pending)))))
  END
END Assertion;
```

```
PROCEDURE Wait():BOOLEAN;
BEGIN
  RETURN Empty(Sender.MessageToSend) OR
     (NOT Receiver.UserWishesToReceive)
END Wait;

BEGIN
  m:=MakeMessage(1);
  Sender.MessageToSend:=Add(Sender.MessageToSend,ADDRESS(m));
  Sender.MessageToSend:=Add(Sender.MessageToSend,ADDRESS(m));
  Sender.MessageToSend:=Add(Sender.MessageToSend,ADDRESS(m));
  Receiver.UserWishesToReceive:=TRUE;
END  AB.
```

A TANULMÁNYOK SOROZATBAN 1982-BEN MEGJELENTEK

130/1982 Barabás Miklós - Tőkés Szabolcs: A lézer printer képalkotás hibái és optikai korrekciójuk

131/1982 RG-II/KNVVT "Szisztemü upravlenija bazani dannüh i informacionnüe szisztemü"
Szbornik naucsno-iszszledovatel'szkih rabot rabocsej gruppü RG-II KNVVT, Bp. 1979. T o m I.

132/1982 RG-II/KNVVT T o m I I.

133/1982 RG-II/KNVVT T o m I I I.

134/1982 Knuth Előd - Rónyai Lajos: Az SDLA/SET adatbázis lekérdező nyelv alapjai (orosz nyelvü)

135/1982 Néhány feladat a tervezés-automatizálás területéről. Örmény-magyar közös cikkgyüjtemény

136/1982 Somló János: Forgácsoló megmunkálások folyamatainak optimálási és irányitási problémái

137/1982 KGST I-15.1. Szakbizottság 1979. és 80. évi előadásai

MSZH Nyomda