Eötvös Loránd Tudományegyetem

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Membránrendszerek vizualizációja és szimulálása

Témavezető:
Kolonits Gábor
egyetemi tanársegéd

Szerző:
Buzga Levente
programtervező informatikus BSc

Budapest, 2022

A membránrendszer vagy más néven P-rendszer olyan biológiailag inspirált számítási modell, amely képes a biokémiai folyamatok törvényszerűségeinek modellezésére. A rendszer több belső térből, úgynevezett régióból áll, amelyek hierarchikus membránstruktúrát alkotnak. Ez lehetővé teszi a sejten belüli és a sejtek közötti információáramlás formális modellezését.

Az egyes régiókban különböző nyersanyagok fordulhatnak elő, amelyek egymással reakcióba léphetnek. A reakció következtében ezek a nyersanyagok átalakulhatnak, régiók között átvándorolhatnak, illetve membránok is feloldódhatnak. Ezeket a reakciókat evolúciós szabályoknak hívjuk. A felépítésükből adódóan a membránrendszerek nagy mértékű párhuzamosításra is alkalmasak. A membránrendszer számítása közben kialakuló állapotát egy konfigurációja írja le, amelyet minden membrán esetén objektumokból álló multihalmaz reprezentál. A számítás eredményét többféleképpen is definiálhatjuk, alapértelmezetten egy kitüntetett membránban a megálláskor jelen lévő objektumok mennyiségét jelenti.

Szakdolgozatom célja egy olyan szoftver elkészítése, amely képes különböző típusú membránrendszerek grafikus megjelenítésére, betöltésére és exportálására, illetve egy megadott szabályrendszer mentén történő számításaik szimulálására. A felhasználónak lehetősége van a membránstruktúra vizuális megkonstruálására, illetve egy megadott formátum segítségével szabályok rögzítésére. A számítás végrehajtásánál támogatott a lépésenkénti futtatás.

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. A számításelmélet rövid története

A számításelmélet az 1930-as évek közepén Alan Turing és Alonzo Church munkássága révén vált külön tudományággá, amelynek központjában a számítások és a számítógépek formális modellezése és elemzése áll. A terület azóta is kulcsfontosságú szerepet kap az informatikában. Az előbb említett matematikai logikával foglalkozó tudósok és társaik arra a kérdésre szerettek volna választ adni, hogy mit jelent maga a számítás? Ezzel az algoritmus matematikai modelljének megalkotását tűzték ki célul. Természetesen a számításelmélet létrejötte előtt is születtek algoritmusok, de azok nem törekedtek formalitásra, hanem egyszerűen csak rögzítették a végrehajtandó utasítások sorrendjét. A kérdés megválaszolása érdekében Turing bevezette a Turing gépet, Church pedig a lambda kalkulust. Ezen eszközök segítségével elkezdték vizsgálni a kiszámíthatóságot. Azt fedezték fel, hogy vannak olyan számítási feladatok, amelyekre nem adható algoritmikusan válasz, illetve, hogy vannak olyan univerzális számítógépek, amelyek minden másik számításra alkalmas gépet szimulálni tudnak. Ez vezetett a Church-Turing tézis megfogalmazásához, amely szerint minden olyan matematikailag formalizálható probléma, ami megoldható valamilyen algoritmussal, az Turing géppel is megoldható. Később kiderült, hogy léteznek a Turing géppel azonos számítási erejű modellek, viszont azóta sem találtak ennél nagyobb számítási erővel rendelkező absztrakt modellt. Ezek szerint ha elfogadjuk a tézist, akkor a Turing gép tekinthető az algoritmus formális modelljének.

A tudományág fejlődését ösztönözte és elősegítette a digitális számítógépek megjelenése. A tézis után pár évvel Konrad Zuse elkészítette az első általános célú progra-

mozható elektromechanikus számítógépet, amelyet elektromágneses relékkel működtetett. 1945-ben készült el az első teljesen elektromos számítógép, az ENIAC, majd Neumann János 1947-ben megfogalmazta a mai számítógép működésének alapjául szolgáló Neumann-elveket. Azóta a számítógépek számítási kapacitása rohamos ütemben növekedik, ám az akkor fontosnak vélt számítási elemzések és korlátok a mai napig fontos szempontot adnak algoritmusok megalkotásánál.

A mai számításelmélet nagyon változazos témák tárházával rendelkezik, magába foglalja az algoritmusok, adatszerkezetek, elosztott és párhuzamosított számítások, kvantumszámítások, biológiai számítások vizsgálatát, illetve nem utolsó sorban a bonyolultságelméletet és az információelméletet. Mindezekből látszik, hogy milyen szerteágazó tematikája és problémaköre van a számításelméletnek és hogy ezek az irányvonalak folyamatosan képesek fejlődni és gazdagodni.

1.2. Membránrendszerek

A 20. század második felében egyre nagyobb figyelmet kaptak az olyan számítások, amely a természetben vagy valamilyen természeti jelenséghez kapcsolódóan mennek végbe. Ezen megfigyelések eredményeképpen megszületett a biológiai számítások területe, amelynek számos új módszert és algoritmust köszönhetünk. Ide tartoznak például a genetikus algoritmusok (általánosabban véve az evolúciós algoritmusok), illetve a neurális hálók. Ezen irányvonal kiterjesztésének tekinthetőek az 21.század legelejére kibontakozó területek mint a membránszámítások és a DNS számítások, amelyek közül az utóbbit a molekuláris biológia alapjaira modelleztek.

A membránszámítások legnagyobb inspirációját a biológiai sejtek jelentik, amelyek az élet legelemibb építőelemei. A sejtek működésében és struktúrájában pedig elengedhetetlenek a membránok, amelyek a sejten belüli rétegződést határozzák meg. Egy sejtet a környezetétől a legkülső membránja (úgynevezett skin) határol el, a rétegződésért pedig a belső membránok felelősek. Egy-egy ilyen membrán pedig egy saját belső teret, *régiót* határol el. Ezen régiókban pedig kémiai reakciók mehetnek végbe, amely hatására különböző molekulák átvándorolhatnak a membránokon keresztül új régiókba. Ezzel a térbeli struktúráltsággal egészítik ki a membránszámítások a DNS számításokat.A membránszámítás motivációja, hogy egy olyan modellt tudjuk konstruálni, amely képes a sejtek közötti és sejten belüli molekulák által közvetített információáramlás mechanizmusának leírására, szemléltetésére.

Ezen célkitűzéssel vezette be Gheorge Paun a membránrendszereket 2000-ben. Később az ilyen modelleket P-rendszereknek is elnevezték az ő tiszteletére. Ahhoz, hogy a biokémiai folyamatokat modellezni tudja, figyelembe vette ezen folyamatok törvényszerűségeit. Nevezetesen, hogy az ilyen reakciók végbemeneteléhez kellő számban szükség van a reakcióhoz szükséges kezdeti molekulákra és ha ezek rendelkezésre állnak, akkor a reakció biztosan meg is fog történni. Emellett egyszerre több reakció (vagy akár egy reakció többször) is végbemehet, ha van hozzá(juk) elegendő nyersanyag. Speciális esetekben egy reakció hatására membránok feloldódhatnak, ilyenkor a benne rejlő molekulák és belső régiók az őt körülvevő régióba kerülnek, illetve az is előfordulhat, hogy a reakciók között valamilyen prioritási sorrend is fennállhat; azaz az egyik reakció nem hajtódhat végre, amíg egy a másik reakciónak lehetősége van rá. Ezen reakciók úgynevezett *evolúciós lépésekbe* szerveződnek, amelyek során a régiók molekulái és a membránrendszer struktúrája is átalakulhat.

A számítás központi szereplői az előbbi tulajdonságokkal rendelkező reakciók, amelyek a membránrendszerekben szabályoknak nevezünk, a résztvevő molekulákat pedig objektumoknak. Ezek a szabályok régiókhoz vannak rendelve. A szabályok alkalmazására a maximális párhuzamosság elve érvényes, azaz ha a jelenleg alkalmazott szabályok után még alkalmazható lenne egy újabb vagy egy eddig már alkalmazott szabály, akkor az a szabály végre is lesz hajtva az adott evolúciós lépésben. Ennek megfelelően a membránrendszer evolúciós lépések sorozatán megy keresztül, egészen addig, amíg már nincsen egyetlen alkalmazható szabály sem. Ilyenkor a számítás eredményét alapértelmezetten a környezetbe kijutó objektumok számának tekinjük, de ez egyes rendszerekben másképpen is megadható (például egy kijelölt kimeneti régió objektumainak számával).

Érdeklődésemet a téma iránt az SAT eldöntési probléma aktív P-rendszerekkel (amely típust a dolgozatban nem részletezem) való lineáris időben történő megoldása keltette fel. Ez elsőre egy hihetetlen eredmény, hiszen ez az egyik legismertebb NP-teljes probléma. Az ellentmondást a membránrendszerek masszív párhuzamosítási képességével lehet megmagyarázni, amelynek segítségével a változók és a klózok számának függvényében igaz, hogy lineáris időben leáll, viszont ezt a membránrendszert szimuláló Turing gép már nem tudna minden bemeneti példányra lineáris időben leállni. Ezután kezdtem el foglalkozni membránrendszerekkel és úgy éreztem, hogy egy szimulációs szoftver hasznosnak bizonyulna kutatási és oktatási célokra egyaránt.

1.3. Megvalósítandó feladat

A dolgozatom célja egy olyan szoftver elkészítése, amely lehetővé teszi a felhasználó számára különböző felépítésű membránrendszerek megkonstruálását, grafikus módon való megjelenítését, illetve számításainak végrehajtását. A támogatott számítási modellek magukba foglalják a membránrendszerek alapmodelljét (amelyben egyes régiók feloldódhatnak, illetve a szabályok között lehet prioritási sorrend) és a szimport-antiport rendszereket. A membránrendszer létrehozását a felhasználó egy (megfelelő formátumú) karakterlánc megadásával kezdeményezheti, ami egyszerűen és szemléletesen írja le a membránrendszer struktúráját és opcionálisan tartalmazhatja régiokra bontva a benne található objektumokat. A megalkotott membránrendszer ilyenkor még nem rendelkezik evolúciós szabályokkal, azokat egy külön menü segítségével tudja beállítani. Ilyenkor a kiválasztott régió objektumai is szerkeszthetőek.

A szoftver különösen hasznos tud lenni oktatási célokra, hiszen biztosítja a szabályok elemzésének lehetőségét a lépésenkénti futtatás funkcióval, illetve a teljes szimuláció során megkapott eredmények segítségével tanulmányozható, hogy a létrehozott modell a kívánt nyelvet (ebben az esetben a természetes számok valamely részhalmazát) generálja-e, ezzel segítve a helyes és helytelen működés (nem teljeskörű) kiszűrését. A program biztosít a felhasználó számára egy használati útmutatót, melyben megtalálja a legfontosabb információkat ahhoz, hogy korábbi ismeretek nélkül is képes legyen egy működő membránrendszert megalkotni és számítását szimulálni.

Lehetőség van a membránrendszer aktuális állapotának lementésére, illetve egy korábbi mentés betöltésére. A membránrendszer *JSON* formátumban kerül mentésre, így azt a felhasználónak lehetősége nyílik arra, hogy a fájlban is módosítson, majd a módosított állapotot töltse be. Ezáltal a szoftver biztosítani tudja azt, hogy egy bonyolult vagy gyakran vizsgált membránrendszert nem szükségszerű a kezdetektől megalkotni, elegendő egyszer a kívánt konfigurációját előállítani, majd később betölteni a már elmentett verziót.

A feladatot *Python* programozási nyelven, a *Qt* keretrendszer felhasználásával valósítottam meg.

2. fejezet

Felhasználói dokumentáció

2.1. Az alkalmazás célja

Az alkalmazás célja, hogy segítségével a felhasználó membránrendszereket hozzon létre majd szimulálja számításaikat. A membránrendszer egy olyan biológiailag inspirált számítási modell, amely az eukarióta sejtek működését és felépítését követve evolúciós lépéseken keresztül történő információáramlást ír le membránok között. Minden membrán által körbezárt *ún.* régió tartalmaz evolúciós szabályokat, amelyek nem változnak a membránrendszer működése közben. Az információt a rendszerben a régiókban található molekulák, *ún.* objektumok hordozzák. Egy szabály csak akkor tud végbemenni, ha rendelkezésre állnak a szükséges objektumok kellő számban. Ilyen helyzetekben a szabályoknak végre is kell hajtódnia, tehát nem fordulhat elő, hogy minden objektum hozzáférhető, de nem kerül a szabály alkalmazásra. Egy evolúciós lépésben a maximális párhuzamosság elve érvényesül, azaz a szabályok véletszerűen kerülnek kiválasztása, egészen addig, amíg van alkalmazható szabály. Az egyik szimulálható típusú rendszerben megadhatóak olyan speciális szabályok, amelyek alkalmazásának hatására egy membrán feloldódhat, ilyenkor teljes tartalma (benne levő objektumok és régiók) az őt körbevevő régióba kerül. A szabályok között prioritási sorrend is felállítható. A számítás legfontosabb tulajdonsága annak kimenete, amely általában a legkülső régión kívülre (azaz a környezetbe) kijutó objektumok számát jelenti.

2.2. Hardver és szoftver követelmények

A szoftver futtatásához Linux környezetre van szükség, amely támogatja az `ELF` formátumú bináris állományok értelmezését. Ezen felül a futtatási környezetnek rendelkeznie kell *Python interpreterrel*, illetve a *Qt* keretrendszerhez való hozzáférés érdekében `PySide6` modullal. Az utóbbi könnyen megtehető shell környezetben a `pip install PySide6` paranccsal . A program teljes funkcionalitásának kihasználásához a felhasználó számítógépének a bemeneti perifériák közül egérrel és billentyűzettel kell rendelkeznie. A szoftver hardverigénye nem igényel részletesebb specifikációt.

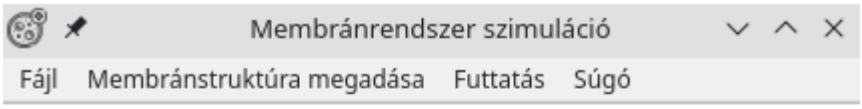
2.3. Futtatás

Mivel a program futtatható állományban kerül a felhasználóhoz, ezért annak az indításhoz elegendő megnyitni a fájlt tartalmazó mappát, majd duplán kattintani a fájlt reprezentáló ikonra. Ugyanez parancssori környezetben is elvégezhető, ilyenkor a terminálban a megfelelő mappába való elnavigálás után a `./MembraneSimulator` parancs megadásával futtatható a program.

2.4. Grafikus felhasználói felület

A felhasználó az alkalmazással a grafikus felhasználói felületen keresztül tud kommunikálni, amely a főablakot és az igény szerint megjelenő dialógusablakokat foglalja magába.

2.4.1. Főablak



2.1. ábra. A főablak az alkalmazás megnyitásakor

A főablak az alkalmazás megnyitásakor még tartalmaz egyetlen grafikus elemet sem, viszont a menüsorban található menüpontok segítségével könnyedén változtatni lehet ezen. Ha a felhasználónak nincs korábbi tapasztala a program használatával, akkor érdemes a *Súgó* menüpont kiválasztásával kezdenie, amelyről részletesen szó esik a 2.5 fejezetben.

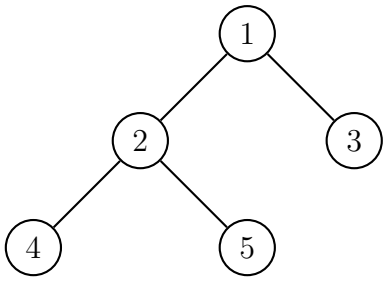
2.4.2. Dialógusablakok

A legfontosabb interakció a felhasználó és a szoftver között a dialógisablakokon keresztül történik.

Membránstruktúra megadása

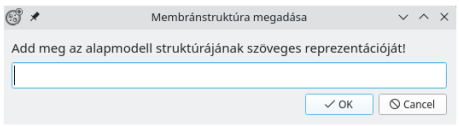
Egy membránrendszer megalkotásának kezdeti módja a struktúrájának megadásával kezdődik. Mivel egy membránrendszerben a membránok hierarchikusan helyezkednek el, ezért a teljes rétegződést nagyon jól lehet ábrázolni fa alakban, ahol mindenkinek a szülő csúcsa az őt legszűkebben tartalmazó régió.

Ezzel egyenértékű az a felírás, amikor egyetlen karakterláncban fejezzük ki ugyanezt, azáltal, hogy egy régiót megfeleltetünk egy nyitó-csukó zárójelpárral. Ilyenkor a két zárójel között elhelyezett objektumok jelentik a régió tartalmát. Azonban nem

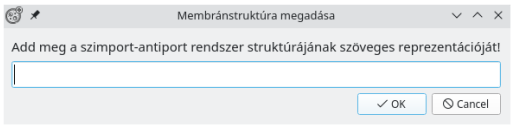


2.3. ábra. Membránrendszer hierarchikus felépítése[1]

csak objektumok, de más régiók is helyet kaphatnak, ezzel kifejezve azt, hogy a már említett régió közvetlen gyerekét szeretnénk megadni.



(a) Dialógusablak alapmodell létrehozásához



(b) Dialógusablak szimport-antiport rendszer létrehozásához

2.2. ábra. Dialógusablakok membránrendszer létrehozásához

A 2.2 ábra a különböző típusú membránrendszerek struktúrájának megadásához használt dialógusablakot mutatja. A megadott karakterláncban a régiók kezdetét és végét jelző zárójelpárok, illetve a bennük előforduló objektumok szerepelhetnek. A helyes formátum feltétele, hogy a zárójelpárok karakterein és az szimport-antiport rendszereknél a kimeneti régió jelzésére használt speciális # karakteren kívül csak az angol ábécé kisbetűi szerepelhetnek a bemenetben, illetve annak meg kell felelnie a helyes zárójelezés szabályainak. Ezt azt jelenti, hogy nem lehetnek átfedések régiók között, azaz olyan karakterláncok, amikor egy külső régió hamarabb kerül lezárásra, mint bármelyik benne lévő régió. Ha ezen feltételeknek megfelel a felhasználó által megadott karakterlánc, akkor a főablak teljes területét lefedő vásznon megjelenik a létrehozott membránrendszer.

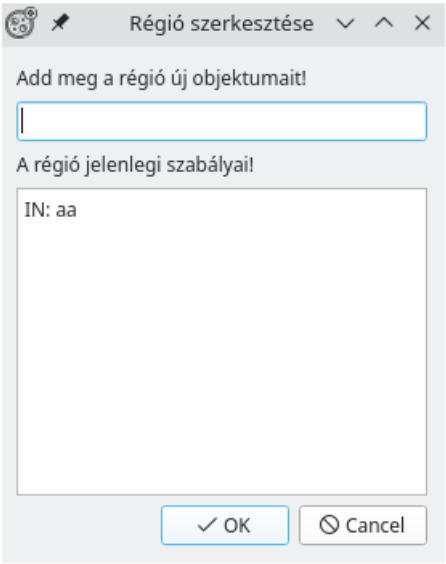
Ha a megadott karakterlánc nem a megfelelő formátumú, akkor a felhasználó figyelmeztetésére egy hibaüzenet jelenik meg a képernyőn.

Objektumok módosítása

Miután a felhasználó megkonstruálta a membránrendszer szerkezetét, utána lehetősége nyílik arra, hogy az egyes régióinak tartalmát szerkessze. Mivel minden egyes

[1]Az ábrán látható struktúra szöveges megfelelője a $[_1[_2[_4]_4[_5]_5]_2[_3]_3]_1$ karakterlánc

régióhoz tartozhatnak objektumok és szabályok is egyaránt, ezért ezek módosítása közös dialógusablakban hajtható végre. A dialógusablak megjelenése a szerkeszteni kívánt régió által lefedett területre történő dupla kattintással kezdeményezhető.

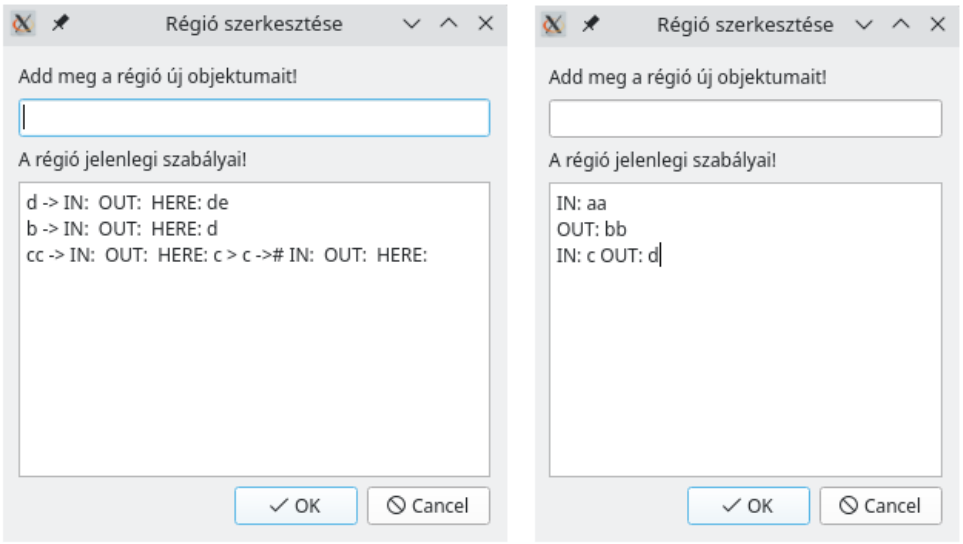


2.4. ábra. Dialógusablak egy régió tartalmának szerkesztéséhez

A régió objektumai közé tetszőleges számban írható a szóköz, mint elválasztó karakter, amely nem kerül figyelembevételre a régió új objektumainak feldolgozásakor. A megengedett karakterek ezen felül továbbra is az angol ábécé kisbetűi. Ha az adott régió a dialógusablak megnyitásakor már rendelkezik objektumokkal, akkor azok alapértelmezett szövegként fognak szerepelni a szövegdobozban. Tehát ha a régió szerkesztésekor az a felhasználó célja, hogy a kiválasztott régió tartalmát újabb objektumokkal egészítse ki, abban az esetben elegendő a hozzáadni kívánt karakterek begépelése.

Szabályok módosítása

A 2.2 segítségével alkotott membránrendszerek még nem fognak szabályokat tartalmazni, ám ezek nélkül nem beszélhetünk még szimulációról. A szabályok fontos szerepet kapnak az információáramlás vezérlésében, hiszen a hatásukra tudnak objektumok átalakulni, be- és kivándorolni a régiók között.



(a) Régió szabályainak szerkesztése alapmodellben

(b) Régió szabályainak szerkesztése szimport-antiport rendszerben

2.5. ábra. Membránrendszerek különböző formátumú szabályokkal

A szabályok felépítése függ a membránrendszer típusától. Minden szabály megadása során először fel kell tüntetni, hogy mik azok az objektumok (olyankor ugyanolyan objektumból akár több is egyszerre), amelyek szükségesek a szabály alkalmazásához.

Szimport-antiport rendszereknél ez a feltétel teljes mértékben elegendő ahhoz, hogy a szabály létrejöjjön, hiszen abban a modellben nincs lehetőség új objektumok létrehozására; a hangsúly inkább az objektumok régiók közötti mozgatására helyeződik. Ahogyan a 2.5b alsó szövegdobozában is látható, szimport-antiport rendszerek esetében két különböző típusú szabály hozható létre. Az első a szimport szabály, amelynek két formája van. Az *IN:* kezdetű szabály alkalmazása esetén a kijelölt objektumok a membránt körülvevő tartományból a jelenleg szerkesztés alatt álló tartományba vándorolnak; az *OUT:* kezdetű szabálynál pedig a szerkesztés alatt álló régióból a membránt körülvevő tartományba kerülnek az objektumok. A másik típushoz az ún. *antiport szabályokat* soroljuk, amelynél egyszerre történik meg az előbbiekkel analóg módon be- és kivándorlás. A bevándorló objektumok előtt az *IN:* prefix szerepel, a kivándorló objektumok pedig az *OUT:* után találhatóak. Ezen két rész a karakterláncon belül felcserélhető, illetve köztük és az objektumok között tetszőleges számú szóköz karakter szerepelhet, amelyek a szabály megalkotásakor nem játszanak szerepet.

Ahogy a 2.5a ábra is mutatja, az alapmodell esetében a szabályok komplexebb felépítéssel rendelkezhetnek. Első nagy különbség, hogy ebben a modellben objektumok átalakulhatnak, illetve eltűnhetnek vagy megsokszorozódhatnak. Ebben a modellben a szabály alkalmazásához szükséges objektumokat a szabály *bal oldalának* nevezzük. A szabály meghatározása ezen objektumok felírásával kezdődik. A szabály alkalmazásának köszönhetően létrejövő objektumok halmazát a nevezzük a szabály *jobb oldalának*, amelyben az objektumok mozgásának figyelembevételével 3 csoportra bonthatunk:

1. Kivándorló objektumok, amelyek a szülő régióba fognak jutni. Ha a legkülső régióban alkalmaztuk a szabályt, akkor az ilyen címkéjű objektumok a környezetbe kerülnek. A szabály megalkotásánál ezen objektumok elé az *OUT:* prefix kerül.
2. Bevándorló objektumok, amelyek véletlenszerűen valamelyik gyerek régióba fognak kerülni (az alapmodell ennél specifikusabban, címke alapján is megengedi az objektumok mozgását, ezt azonban az alkalmazás nem támogatja). A szöveges formátumban ezen objektumokat az *IN:* szöveg előzi meg.
3. Régión belül létrejövő objektumok, amelyek a szabályhoz tartozó régióba kerülnek. Ezen objektumok előtt a karakterláncban a *HERE:* kifejezés kell álljon.

A szabály bal és jobb oldalát a *"->"* szimbólumok választják el, amelyek együttesen egy nyilat reprezentálnak, annak kifejezésére, hogy a bal oldalt a jobb oldal objektumai „váltják fel". Tehát ekkor a *"aa -> IN: bb OUT: cc HERE: dd"* szöveghez tartozó alapmodellbeli szabály két a objektumot igényel a végbemeneteléhez (amelyek felemésztődnek a reakció hatására), a régióból kivándorol két b objektum, az egyik gyerek régióba bevándorol két c objektum, illetve régión belül kialakul két új d objektum.

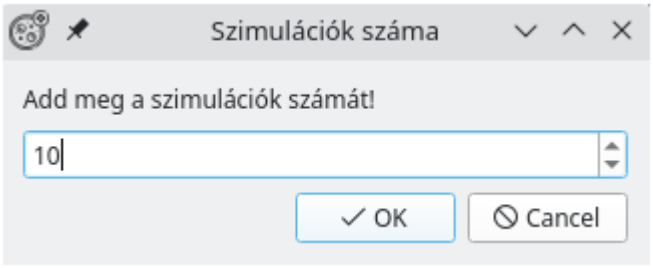
Az alapmodell szabályai kiegészülhetnek többletjelentéssel is. Ebben a modellben ugyanis megengedett lehet egy régió felbomlása, amely egy vagy egyszerre több szabály előidézésére is bekövetkezhet. Ez a szándék jelezhető egy szabály konstruálásakor, ha a bal- és jobb oldalt elválasztó nyilat reprezentáló karakter után közvetlenül egy # karaktert ír a felhasználó. Ilyenkor a régió tartalma és gyerek régiói a szülő régiójába kerülnek.

Szimulációk számának megadása

A korábbi dialógusablakok segítségével a felhasználó lehetőséget szerzett arra, hogy a teljes mértékben a saját elképzeléseinek megfelelő membránrendszer hozza létre és konfigurálja fel objektumokkal és szabályokkal. Ezek után már csak a rendszer számításának szimulálása van hátra. A szimulálás végrehajtására két módot biztosít az alkalmazás:

1. Teljes szimuláció futtatása párhuzamosan tetszőleges számú másolattal
2. Egy szimulációs lépés az aktuális membránrendszeren

Fontos megjegyezni, hogy a teljes szimuláció futtatásának kiválasztása esetén a képernyőn látható membránrendszer nem kerül szimulálásra, annak érdekében, hogy a szimulálni kívánt állapot megőrződjön, így utána változtatások nélkül kezdeményezhető a funkció megismétlése.

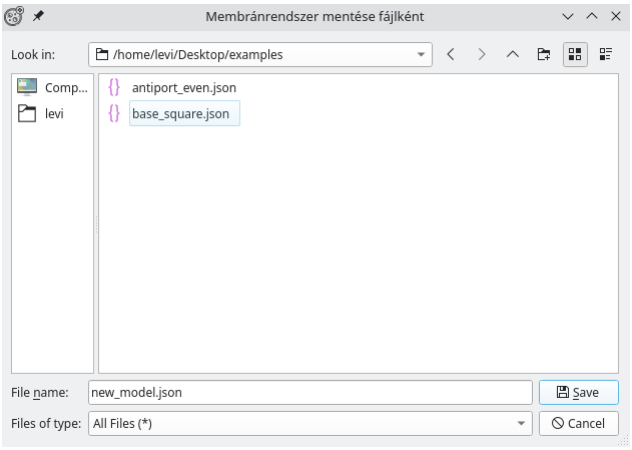


2.6. ábra. Dialógusablak a szimulációk számának kiválasztásához

A szimuláció ismétléseinek számát egy olyan speciális szövegdobozzal lehet beállítani, amely bemenetként csak numerikus értékeket reprezentáló karaktereket fogad el. Alternatívaként a szövegdoboz jobb oldalán található nyilakkal is egyesével állítható az érték, amelynek alsó korlátja *1*, felső korlátja pedig *1000*.

Mentés

Miután a felhasználó kialakított egy olyan membránrendszert, amelyet szeretne elemezni vagy későbbi használat céljából megőrizni, az alkalmazás lehetőséget biztosít a jelenlegi állapot fájlba történő lementésére.



2.7. ábra. Dialógusablak a membránrendszer mentéséhez

A mentés folyamata a *Mentés* gombra történő kattintással vagy a `Ctrl+S` billentyűkombináció lenyomásával kezdeményezhető. A funkció kiválasztása után egy dialógusablak jelenik meg, amelyben a mentés helyére kell navigálnia a fájlrendszerben, majd a fájlnév megadása után az *OK* gomb lenyomására létrejön a *JSON* formátumú fájl a megadott útvonallal.

Mivel a *JSON* formátum könnyen értelmezhető és szerkeszthető, ezért egy jártasabb felhasználója a programnak képes egy lementett membránrendszeren módosításokat végezni a megfelelő szerkesztésekkel.

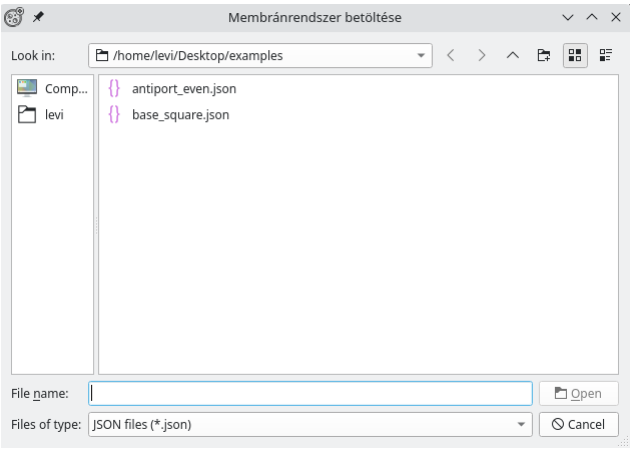
```
{
  "type": "SymportAntiport",
  "structure": "[[[#]]]",
  "rules": {
    "0": [
      "OUT: b",
      "IN: ab"
    ],
    "1": [
      "IN: aa",
      "IN: b OUT: c"
    ],
    "2": [
      "IN: aa"
    ]
  },
  "env_obj": "",
  "env_inf": "a",
  "objects": {
    "0": "b",
    "1": "c",
    "2": ""
  }
}
```

2.8. ábra. Egy lementett membránrendszerhez tartozó JSON fájl [2]

[2] Az ábrán látható konfiguráció a páros számokat generáló szimport-antiport rendszerhez tartozik

Betöltés

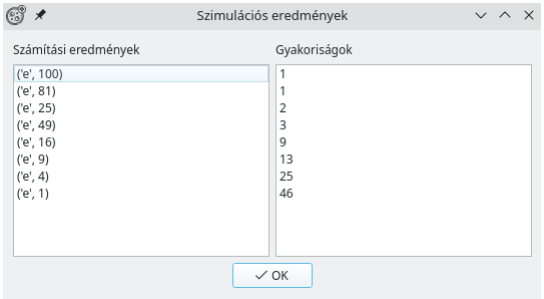
Az alkalmazás biztosítja lementett membránrendszerek betöltését, amelynek hatására betöltődik a kiválasztott fájlhoz tartozó állapot a grafikus felületre. A betöltés nagyon hasonlóan működik a mentéshez, kezdeményezni a *Betöltés* gombra történő kattintással vagy a `Ctrl+O` billentyűkombináció lenyomásával lehetséges, mely után egy dialógusablak segítségével a felhasználó elnavigálhat a megnyitni kívánt fájlt tartalmazó mappához.



2.9. ábra. Dialógusablak egy membránrendszer betöltéséhez

Eredményablak

Miután a felhasználó létrehozott egy membránrendszert és szimulálta azt, nincs más hátra mint értesülnie annak eredményéről. Ezt a funkciót látja el az eredményablak, amely a membránrendszer számításának befejeződése után automatikusan megjelenik és egy listában gyakoriság szerint növekvő sorrendbe rendezve jeleníti meg a különböző számítási eredményeket.



2.10. ábra. Szimulációs eredményeket kilistázó ablak

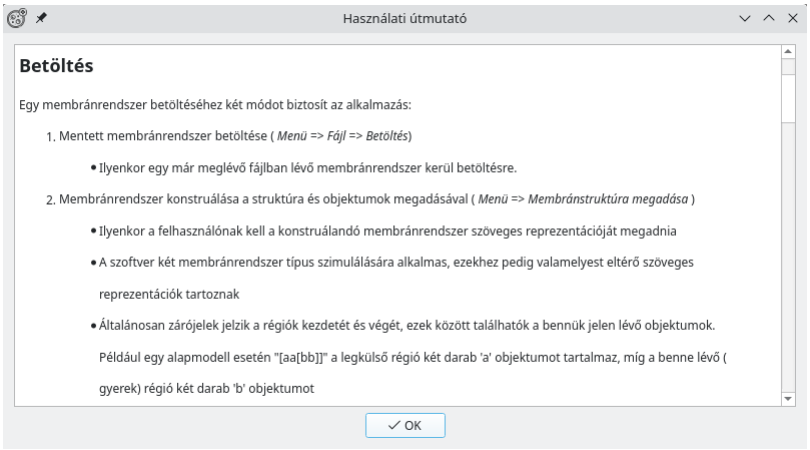
2.5. Használati útmutató

A szoftver használata a membránrendszerek előzetes ismerete nélkül kriptikus tud lenni, ezért a felhasználó útbaigazítására és a szükséges jellemzőkkel kapcsolatos tudnivalók ismertetésére egy súgó funkció is megtalálható a menüsorban.

A *Súgó* gombra való kattintás után jelenik meg az ablak, amelyben a korábbi fejezetekhez hasonló felbontásban találhatóak meg az alkalmazás használatával kapcsolatos információk.

A használati útmutató a *Markdown* formátumnak megfelelő formázásban jelenik meg, amely jól strukturált és könnyen olvasható. A tartalma fejezetekre van bontva, amelyek kitérnek a korábban említett dialógusablakok sajátosságaira, az objektumokkal és szabályokkal kapcsolatos elvárásokra és a szoftver helyes használatának módjára.

A használati útmutató segítségével a felhasználó a membránrendszerekkel kapcsolatos alapvető információkat és a szoftver használatához szükséges tudnivalókat ismerheti meg. A súgó részletes tájékoztatást ad a membránrendszerek létrehozásáról, objektumainak és szabályainak megváltoztatásának módjáról és azok helyes formátumáról, a mentés és betöltés folyamatáról, illetve a számítási eredmény értelmezéséről.



2.11. ábra. Használati útmutató ablak

3. fejezet

Fejlesztői dokumentáció

A szoftver működésének részletezéséhez elengedhetetlen a membránrendszerek formális modelljének ismerete, hiszen ezen rendszerek sajátos viselkedése az architechturális kérdésekben döntő szerepet képez.

A teljes definíció előtt azonban érdemes még pár fogalmat bevezetni. Általánosan tehát egy olyan absztrakt számítási modell áll a szoftver központjában, amely egy biológiai sejthez hasonló felépítéssel rendelkezik, az egymásba ágyazott membránok régiókat különítenek el, amely régiókban különböző objektumok helyezkedhetnek el. A rendszerben az objektumokon felül szabályokat is rendelhetünk régiókhoz, ezen szabályok felelősek a membránrendszer állapotában végbemenő változásokért. A szabályokat a megfelelő feltételek teljesülése esetén kötelesek vagyunk alkalmazni, így egy evolúciós lépés a rendszer számításában addig tart, amíg van legalább egy akalmazható szabály. Érdemes már megjegyezni, hogy egy régió beazonosításához egy egyedi azonosítóra van szükség, illetve, hogy egy régiót könnyen azonosítani lehet az őt legszűkebben tartalmazó membránnal. Így természetes módon adódik, hogy a régiókat fa struktúrába tudjuk szervezni, amelyben minden egyes csúcs egy egyedi azonosítóval rendelkezik. Ezen fában a levelek az elemi membránokat reprezentálják, azaz azokat a régiókat, amelyeken belül nem helyezkedik el régió. Ezen struktúra pedig nagyon könnyen megfeleltethető egy sztringnek.

A másik fontos tervezési szempont az objektumok reprezentálásához kapcsolódik. A membránrendszerek az egy régión belül megtalálható objektumok sokaságát ún. *multihalmazokkal* reprezentálják, amelyben minden objektum rendelkezik egy multiplicitás értékkel, amely számosságát mutatja meg a régión belül. A multihalmaz teljeskörű értelmezéséhez pedig szükség van egy ábécére, amely a lehetsége-

sen előforduló objektumok halmazát tartalmazza. Ezen multihalmazok felett pedig műveleteket kell tudni értelmezni, amelyek az evolúciós szabályok által megkövetelt funkciókat kell magukba foglalniuk. Mivel egy evolúciós szabály felhasználja az alkalmazásához szükséges objektumokat (azaz szintén egy multihalmazt), ezért szükség van egy olyan műveletre, amely a régióhoz tartozó multihalmazból eltávolítja a felhasznált objektumokat. Ezt a funkciót a multihalmazok közötti kivonás művelet fogja biztosítani. Ahhoz, hogy eldöntsük, hogy egy szabály alkalmazható-e egy régióban, elegendő megvizsgálni, hogy a régióhoz tartozó multihalmaz magába fogalalja-e a szabályhoz tartozó bal oldali objektumok multihalmazát. Erre a problémára a halmazok közötti részhalmaz reláció nyújt megoldást. Végül az újonnan keletkező objektumokból álló multihalmaz hozzáadását a régió jelenlegi tartalmához a multihalmazokra nézett unió művelettel lehet modellezni.

Ezen megjegyzések után már bevezethetjük a membránrendszerek formális definícióját.

3.1. Definíciók, formális modell

1. Definíció. Egy $\Pi = \langle O, \mu, \omega_1, \ldots, \omega_m, R_1, \ldots, R_m, i_o \rangle$ rendezett $(2m + 3)$-ast membránrendszernek (alapmodell) nevezzük, ha

1. O egy objektumokból álló ábécé
2. μ egy m membránból álló membránstruktúra. A régiók ekkor a $\{1, 2, \ldots, m\}$ elemeivel injektív módon vannak címkézve. Ilyenkor m-et Π fokának nevezzük.
3. $\omega_1, \ldots, \omega_m$ O feletti multihalmazokat reprezentáló sztringek, amelyek rendre az $1, 2, \ldots, m$ címkéjű régiókhoz vannak rendelve
4. $R_i, 1 \leq i \leq m$ μ i-edik membránjához rendelt O feletti *evolúciós szabályok* véges halmaza. A szabályok $u \to v$ alakúak, ahol $u \in O^+, v \in (O \times TAR)^*$, ahol $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$ Ha nem írjuk ki a j indexet, akkor nemdeterminisztikusan történik régió kiválasztása
5. $i_o \in \{1, 2, \ldots, m\}$ egy elemi membrán címkéje, amely a modell kimeneti membránja.

Az előbbi definícióban meghatározott membránrendszerre a dolgozatban alapmodellként fogok hivatkozni. Az alkalmazás az alapmodell esetén csak a nemdeterminisztikus *in* címkét engedi meg a szabályoknál. Ennek oka, hogy a szöveges reprezentációban csak így garantálható az, hogy egy karakter egy objektumot reprezentál, bárhol is szerepeljen a sztringben, illetve a régiók kézzel történő felcímkézésére sincs ezáltal szükség, hiszen ez lenne az egyetlen művelet, amely explicit használja a címkéket.

A számítás formalizálásához fontos bevezetni a konfiguráció fogalmát.

2. Definíció. A $C = (w_1, \ldots, w_m)$ *a* $\Pi = \langle O, \mu, \omega_1, \ldots, \omega_m, R_1, \ldots, R_m, i_o \rangle$ membránrendszer konfigurációja, ha $w_i \in O^*$ és w_i az i-edik régióban lévő objektumokból álló multihalmaz sztring reprezentációja

Megjegyzés. Egy C konfiguráció kezdőkonfiguráció, ha $\forall 1 \leq i \leq m$ esetén $w_i = \omega_i$

3. Definíció. A megállási konfiguráció olyan konfiguráció, amelyre nem lehet már evolúciós szabályt alkalmazni.

4. Definíció. Az egylépéses konfigurációátmenet $C_1 \Longrightarrow_\Pi C_2$, akkor ha C_1-ből egy evolúciós ütemben megkapható C_2 (a maximális párhuzamosság elvének megfelelve).

Megjegyzés. Mivel a szabályok alkalmazása nemdeterminisztikus, ezért a konfigurációátmenet relációban egy C_1-hez több C_2 is létezhet.

5. Definíció. A többlépéses konfigurációátmenet a $\Longrightarrow_\Pi$ reláció reflexív tranzitív lezártja.

A rendszer egy számítása alatt tehát a kezdőkonfigurációból egy megállási konfigurációba történő többlépéses konfigurációátmenet-sorozatot értünk.

Megjegyzés. Ilyenkor a számítás eredményét többféleképpen definiálhatjuk. Alapértelmezetten a kimeneti régióban lévő objektumok számát jelenti. Az alkalmazás az alapmodell esetén ettől eltér és a környezetbe kijutó objektumok számát tekinti a számítás eredményének.

Létezik olyan kiterjesztése az 2 definíciónak, amelyben megengedünk olyan $u \to v\delta$ alakú szabályokat, ahol az $u \to v$ az eddigiekhez hasonló evolúciós szabály, a δ pedig egy speciális szimbólum, amely a membrán feloldódását jelzi a szabály alkalmazásának hatására. Tehát ha egy evolúciós lépésben alkalmazásra kerül egy

ilyen speciális szabály, akkor az ütem végén a szabályhoz tartozó régió felbomlik, ilyenkor a benne lévő objektumok és membránok (a szabályok nem) a membránt közvetlenül tartalmazó szülő régióba kerülnek. A legkülső régió sosem oldódhat fel.

A másik kiterjesztés a szabályok feletti részberendezés lehetőségét adja meg, mely szerint ha r_1 és r_2 relációban állnak (amelyet jelölhetünk $r_1 < r_2$-vel), akkor csak abban az esetben alkalmazható az r_1 szabály, ha r_2 már nem alkalmazható az evolúciós lépésben.

A szoftver mindkét kiterjesztést alapértelmezetten támogatja, működésüket a későbbi fejezetekben részletesebben kifejtem.

Az alkalmazás az alapmodell mellett a szimport-antiport rendszerek szimulálását támogatja. Ebben a modellben az objektumok nem alakulhatnak át, csak a membránstruktúrán belül és a környezetbe vándorolhatnak. Az eddig definiált fogalmak a szimport-antiport rendszereknél is helyt állnak, egyedül a szabályok alakja és működéseigényel módosításokat. Ezen felül a szimport-antiport rendszerek környezete rendelkezhet az O ábécé egy olyan speciális részhalmazával, amelyben előforduló objektumok korlátlanul rendelkezésre állnak az evolúciós lépések során. Ennek megfelelően vezessük be az alábbi definíciót.

6. Definíció. Egy $\Pi = \langle O, \mu, E, \omega_1, \ldots, \omega_m, R_1, \ldots, R_m, i_o \rangle$ rendezett $(2m + 4)$-est szimport-antiport rendszernek nevezünk, ha

1. O egy objektumokból álló ábécé
2. μ egy m membránból álló membránstruktúra. A régiók ekkor a $\{1, 2, \ldots, m\}$ elemeivel injektív módon vannak címkézve. Ilyenkor m-et Π fokának nevezzük.
3. $\omega_1, \ldots, \omega_m$ O feletti multihalmazokat reprezentáló sztringek, amelyek rendre az $1, 2, \ldots, m$ címkéjű régiókhoz vannak rendelve
4. $E \subset O$ a környezetben korlátlanul rendelkezésre álló objektumok halmaza
5. $R_i, 1 \leq i \leq m$ μ i-edik membránjához rendelt O feletti szimport/antiport szabályok véges halmaza
6. $i_o \in \{1, 2, \ldots, m\}$ egy elemi membrán címkéje, amely a modell kimeneti membránja.

Ebben a modellben legyenek $x, y \in O^+$ objektumok multihalmazait reprezentáló sztringek. A velük alkotott szabályok az alábbi alakok egyikét ölthetik:

1. (x, in): Ilyenkor az x multihalmaz a membrán körül elhelyezkedő, őt legszűkebben tartalmazó régióból a membrán által határolt régióna vándorol

2. (x, out): Ilyenkor az x multihalmaz az őt tartalmazó régióból a membránt körülvevő régióba mozog

3. (x, in; y, out): Ilyenkor az x multihalmaz a membrán körül elhelyezkedő, őt legszűkebben tartalmazó régióból a membrán által határolt régióna vándorol, illetve az y multihalmaz a szabályt tartalmazó régióból a membránt körülvevő régióba mozog

Az első két szabályt *szimport szabálynak*, míg az utolsót *antiport szabálynak* nevezzük.

Megjegyzés. Ezen két típuson kívül léteznek még *uniport* szabályok is, amelyek felfoghatók olyan speciális szimport szabályként, amelyekben $|x| = 1$, ahol $|x|$ a multihalmazban szereplő objektumot számát jelöli.

A konfigurációkkal kapcsolatos definíciók megegyeznek az alapmodellben leírtakkal, kiegészítve azt a környezet $O - E$-beli objektumaiból álló ω_0 kezdeti multihalmazt reprezentáló sztringgel, a későbbi konfigurációkban pedig w_0-val.

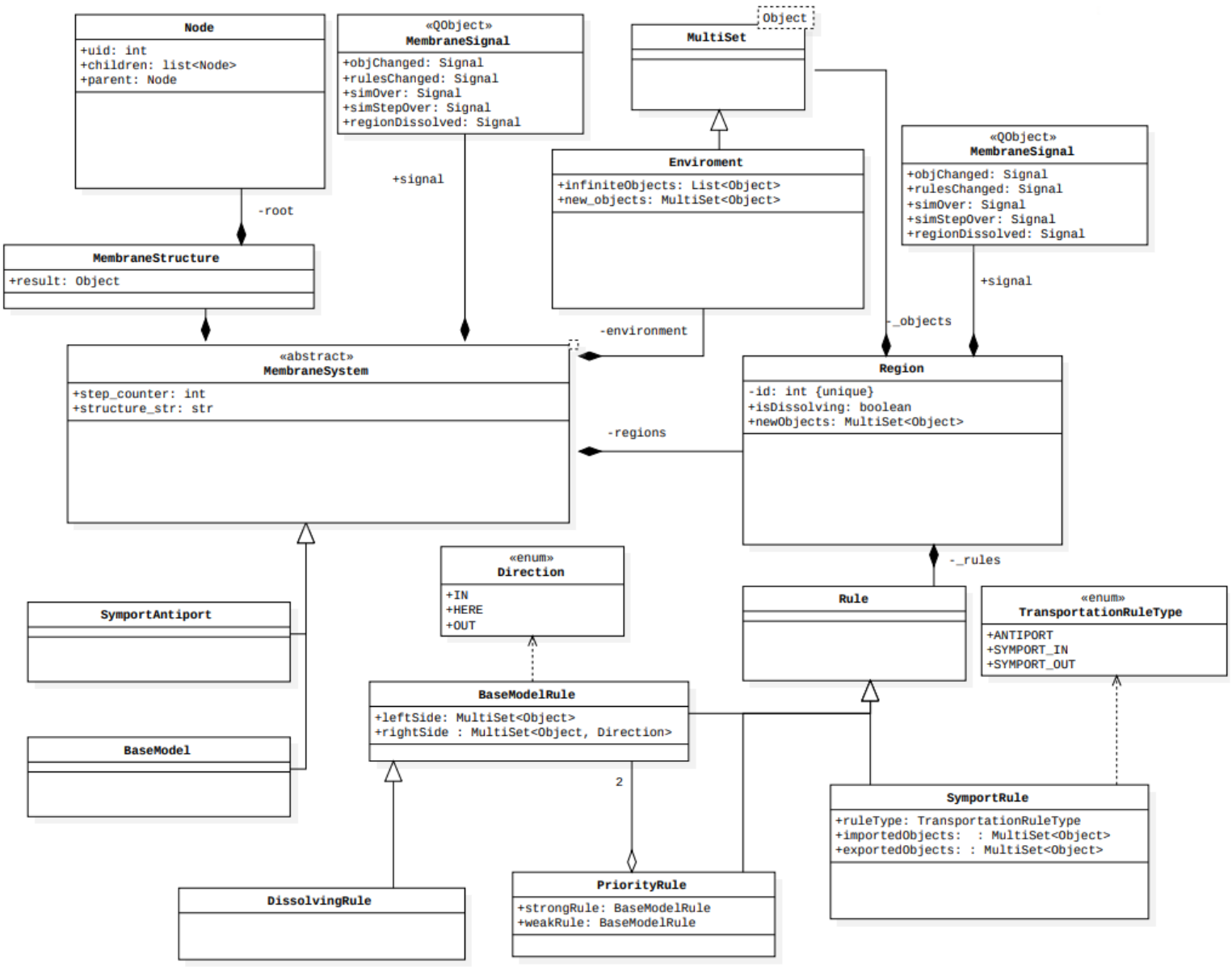
3.2. Tervezés

A feladat modellezésének legelső mozzanata az alkalmazáshoz szükséges osztályok meghatározása. Elsőként megállapítható, hogy a membránrendszerek egyes típusainak sok közös vonása van, ezért célszerű az absztrakt ősosztály `MembraneSystem` bevezetése . Ez az osztály magába foglal olyan műveleteket, amelyet minden szimulálandó membránrendszer köteles implementálni. Ez az osztály a kód karbantarthatóságát és bővíthetőségét is elősegíti, hiszen esetleges újabb membránrendszerek bevezetése esetén elegendő ezen osztály absztrakt metódusait implementálni. A szoftver esetében két konkrét membránrendszer típusról beszélhetünk, az alapmodellről (`BaseModel`), illetve a szimport-antiport (`SymportAntiport`) rendszerekről, amelyek az ősosztály leszármazottaiként vannak jelen az osztály diagramban.

Általánosan a membránrendszerek régiókból állnak, ezért a `MembraneSystem` osztály az egy régiót reprezentáló `Region` osztályból álló gyűjteményt komponál. A két

osztály közötti kapcsolat aggregáció, mivel egy membránrendszer nem létezne régiók nélkül, illetve absztrakt tekintetben egy régió sem létezhet önállóan, hanem egy csak egy komplexebb entitás (eukarióta sejt) részeként. Ezen régiók pedig valamilyen struktúrába szerveződve helyezkednek el a rendszerben, ennek eltárolására és karbantartására elhivatott a `MembraneStructure` osztály, amely a `Node` osztály segítségével egy *n-áris* fát tárol el és a fa bejárásával begyűjti a membránrendszer által igényelt információkat. A régiók jelentős szerepet játszanak a membránrendszer szimulációjában, hiszen a bennük lévő objektumok és szabályok együttesen határozzák meg a generált nyelvet. Az osztály diagram ennek megfelelően a `Region` osztályhoz az objektumok multihalmazát reprezentáló `MultiSet` osztályt és a szabályokat jelölő absztrakt `Rule` osztályból álló gyűjteményt tartalmazza. A szabályokat összefoglaló ősosztályból a konkrét típusokhoz tartozó szabályok származnak le, így az alapmodell esetén `BaseModelRule` osztály képviseli a létrehozható szabályokat, míg a szimport-antiport rendszer esetében a `SymportRule` látja el ezt a feladatot. Az alapmodellben tovább lehet specializálni a szabályokat, hiszen megadhatóak olyan szabályok, amelyek hatására a hozzá tartozó régió felbomlik, ezen funkciót a `DissolvingRule` tölti be, amely a `BaseModelRule`-ból származik le. A másik ilyen kiegészítése az alapmodellnek a szabályok közötti prioritási sorrend lehetősége, amely a `PriorityRule` osztályban aggregált két `BaseModelRule` típusú objektum segítségével éri el a kívánt viselkedést. A szabályok konstruáláskor valahogyan jelezni kell, hogy a kijelölt objektumokat reprezentáló multihalmaz milyen irányultsággal rendelkezik, azaz a szabály alkalmazása után melyik régióba fog kerülni vagy esetleg kijut a környezetbe. Ezt az információt az alapmodellbeli szabályok esetén a `Direction`, szimport-antiport rendszerek esetében pedig a TransportationRuleType (enum) osztály hordozza Jegyezzük meg, hogy nem csak egy régió hordozhat objektumokat, hanem a membránrendszer környezete is, amely tartalmazhat korlátlan mennyiségben is objektumokat. Ezért a membránrendszer a régiókon felül a `Environment` osztályt is komponálja, amely számomtartja a környezetben korlátlan számban rendelkezésre álló objektumok és a kijutó objektumok multihalmazát.

3.3. Modell



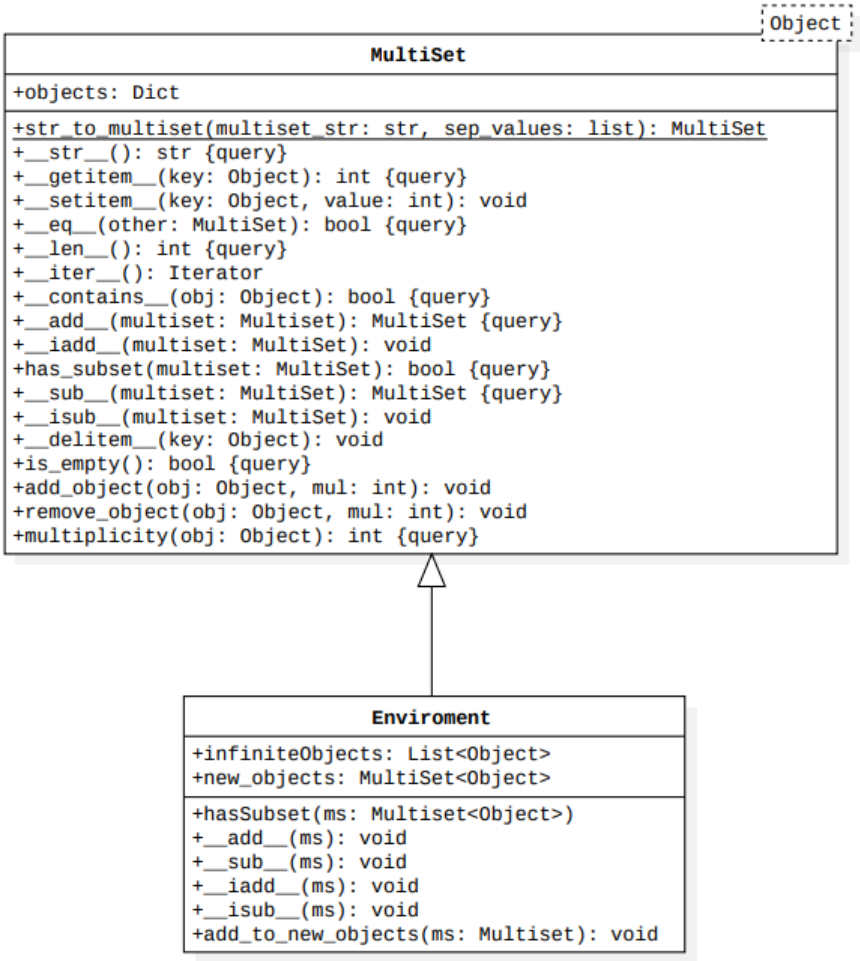
3.1. ábra. A modell osztály diagramja

A 3.1 ábra mutatja az osztály diagramot, amelyben még nem szerepelnek az osztályszintű-és példánymetódusok. Az eddig említett osztályokon felül megjelenik két, a modell és a nézet közötti üzenetváltásért felelős, `QObject`-ből leszármazó osztály, a `MembraneSignal` és a `RegionSignal` . Az előbbi a teljes membránrendszerben előforduló jelzéseket küldi tovább majd a megjelenítésért felelős osztályoknak, míg az utóbbi az egyes régiók tartalmának megváltozását hivatott jelezni. A szoftver működéséhez szükséges objektumok meghatározása még nem elegendő céljának megva-

lósításához, ugyanis szükség van ezen osztályok metódusain keresztül a szimulálás folyamatának megtervezésére is.

3.3.1. Objektumok

Az objektumokat reprezentáló multihalmazokat a `MultiSet` osztály segítségével hozhatjuk létre. Ezen osztály példányai felett értelmezve van az unió, kivonás művelete, a részhalmaz reláció, illetve lekérdezhető egy tetszőleges objektum multiplicitása. A `MultiSet` osztály kiemelt fontosságú a számítás során, hiszen egy szabály alkalmazhatóságának eldöntéséhez két multihalmaz (nevezetesen a régió objektumait tartalmazó és a szabály bal oldala) közötti részhalmaz reláció eldöntésére van szükség. Ha valóban alkalmazható egy szabály, akkor a szabály bal oldala felhasználásra kerül, tehát egy kivonás műveletet kell végezni, a létrejövő objektumok pedig a megfelelő régióhoz az unió műveletével tudnak hozzákerülni. Tehát az osztályt nem csak a régiók, hanem a szabályok is felhasználják működésük során.

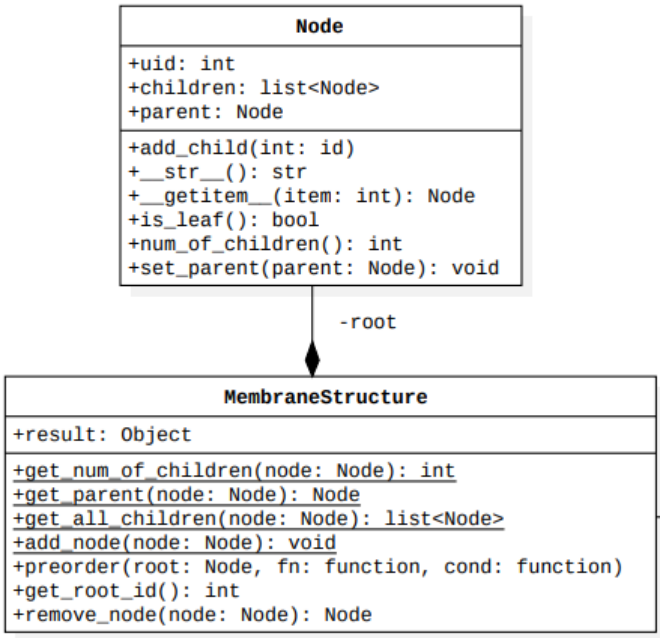


3.2. ábra. Az objektumokat reprezentáló multihalmaz és a környezet

A környezet reprezentálására hivatott `Environment` osztály a korlátlanul rendelkezésre álló objektumokat tartalmazó `infinite_obj` listával egészíti ki a `MultiSet` osztályt, illetve az ősosztályban használt műveletek is felüldefiniálja a helyes viselkedés érdekében. Mivel a környezet is `MultiSet` típusú, ezért a dinamikus típusosság következtében az unió műveletet meghívhatnánk rá `MultiSet` típusú paraméterrel, ennek eredménye szintén `MultiSet` típusú lenne (ami a környezet aktualizálásakor egy evolúciós lépésben nem lenne szerencsés). Ennek elkerülése érdekében ezen műveleteket implementáció nélkül felüldefiniálja az osztályt és csak olyan műveletet biztosít, amely a metódushoz tartozó (`this`) példányhoz unió művelet segítségével hozzáad egy multihalmazt. A hozzáadás során az `infinite_obj`-ban megtalálható objektumok figyelmen kívül hagyhatók. A kivonás esetén is hasonlóan járunk el, illetve ha speciálisan olyan objektumot kell levonni, amely korlátlan számban áll rendelkezésre, olyankor az mellékhatás nélkül végrehajtódhat.

3.3.2. Membránstruktúra

Mivel a membránrendszer struktúrája egyes típusok esetében módosulhat, ezért fontos, hogy a rendszer szerkezetét ne statikusan legyen tárolva, hanem a membránrendszer megkonstruálása után is dinamikusan tudjon változni. Ezeket az elvárásoknak megfelelő n-áris fát láncolt lista adatszerkezettel valósítom meg, amelynek fejelemét (fák esetén inkább gyökerét) a `MembraneStructure` osztály adattagként tárolja el. Minden csúcs egyedi azonosítóval rendelkezik, amely megegyezik a hozzátartozó régió címkéjével. Emellett minden csúcs számontartja a saját szülőjét, illetve gyermekeiből álló `children` listát, amelyhez az `add_child` metódussal lehet új elemet fűzni. Az osztály többi művelete *getter* vagy *setter* metódusként funkcionál.



3.3. ábra. A membránrendszer struktúrájáért felelős osztályok

A `MembraneStructure` osztály a fa struktúra karbantartásáért és a struktúrával kapcsolatos „lekérdezésekért” felelős. Ezen feladatok ellátásához kulcsszerepet kap a `preorder(root, fn, cond)` metódus, amely a bejárja a fát és ha egy n csúcsra teljesül a `cond` függvény feltétele, akkor az `fn` függvény kerül meghívásra.

```
1 def preorder(self, root, fn, cond):
2         if root is None:
3             return
4         if cond(root):
5             self.result = fn(root)
6         else:
7             iter_count = root.num_of_children()
8             for i in range(iter_count):
9                 self.preorder(root.children[i], fn, cond)
```

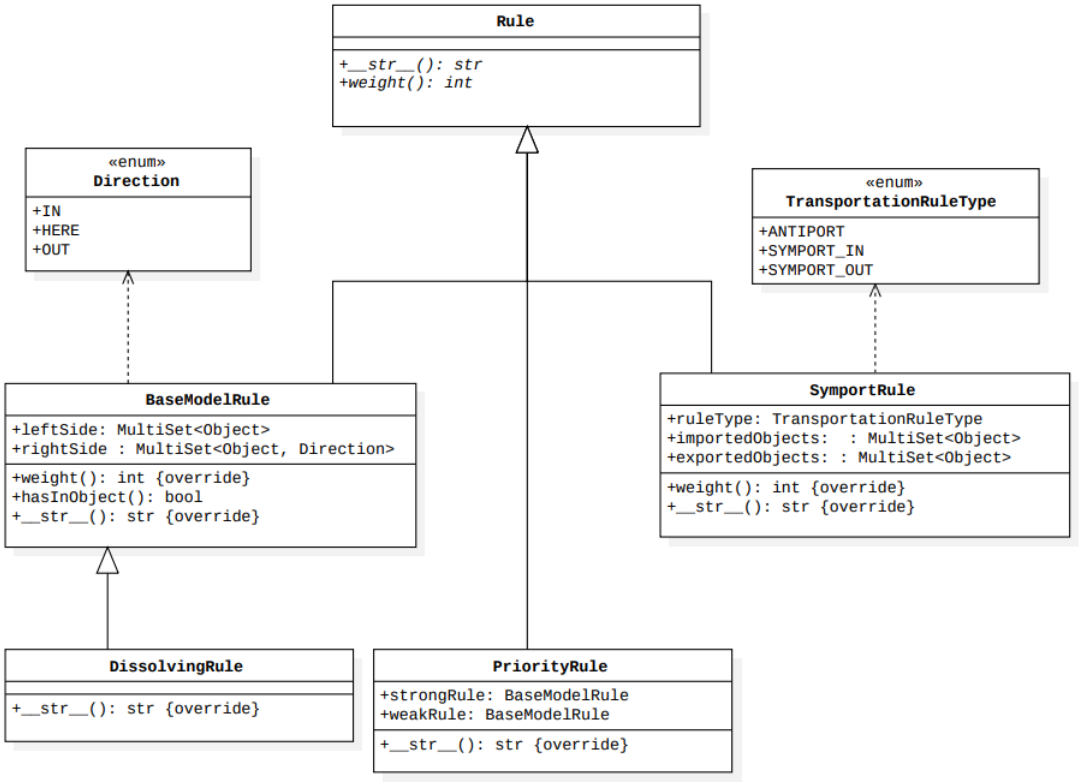
3.1. forráskód. Membránstruktúra preorder bejárása

A `MembraneStructure` kialakítása igazodik ehhez a szemantikához, mivel mind a négy osztályszintű metódus és a `remove_node()` metódus is egy *Node* típusú paramétert vár, amelyen végrehajtják a kívánt műveletet. Tehát ezen metódusok nem járják be a fát és keresik meg a megfelelő címkéjű csúcsot, hanem a `preorder` függvény paramétereként kerülnek felhasználásra, amelyben a feltétel a csúcs egyedi azonosítójára szűr. Mivel a rekurzív függvénynek tetszőleges típusú visszatérési ér-

téke lehetne, ezért a visszaadandó érték a struktúra `result` adattagjába kerül. Az osztály felépítése garantálja, hogy a result lekérdezései nem keresztezhetik egymást, ezáltal elkerülve az inkonzisztens és helytelen értékeket.

3.3.3. Szabályok

A szabályok a membránrendszer számításaiban központi szerepet kapnak, ám kevés olyan tulajdonságuk van, amely minden típusban közös vonásként jelenik meg. Ezért az absztrakt `Rule` ősosztály kevés implementálandó metódust határoz meg, csak a szabályhoz tartozó szöveges reprezentáció, illetve a szabály súlyát meghatározó műveletek meglétét követeli.

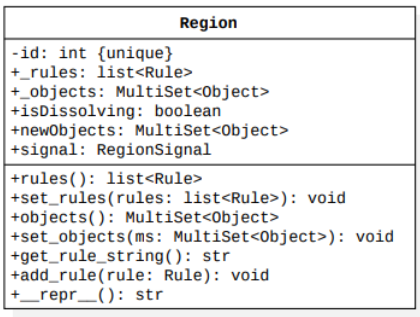


3.4. ábra. A membránrendszer szabályait modellező osztályok

Az alapmodellhez tartozó `BaseModelRule` osztály a leszármazás mellett rögzíti a modellezéséhez szükséges két adattagot. Az egyik a `left_side`, amely a szabály végbemeneteléhez szükséges objektumokból álló multihalmazt tárolja (objektum-multiplicitás párok formájában), a másik pedig a `right_side`, amely a szabály alkalmazásának következtében létrejövő objektumokat és azok mozgásának irányát adja meg. Ezen irányok jelzésére a `Direction` osztályban rögzített `HERE`, `IN` és `OUT` felsorolási típus értékei hivatottak. Tehát a `right_side` olyan multihalmazban a

megfelelő multiplicitással `(objektum, irány)` párok szerepelnek. Tehát egy evolúciós lépésben az alapmodellbeli szabály alkalmazhatóságának eldöntése a szabály bal oldala és a régióhoz tartozó objektumokból álló multihalmaz közötti tartalmazási reláció vizsgálatával ellenőrizhető. Az alapmodell kiegészítéseként beszélhetünk a feloldódás fogalmáról, amely a modellezés során a `BaseModelRule`-ból leszármaztatott `DissolvingRule` osztály segítségével kerül megvalósításra. Ez az osztály semmilyen új metódust vagy adattagot nem vesz fel, az információtartalom a példányosítása mögött rejlik, hiszen ilyen típusú szabály alkalmazása az adott régió `is_dissolving` adattagjának igaz értékre billentését idézi elő. Az alapmodell másik ilyen kiterjesztése a szabályok közötti részbenrendezés lehetőségét nyújtja, amelyet az alkalmazás két szabály közötti prioritás rögzítésével támogatja. Az ilyen szabályok közötti relációk leírására a `PriorityRule` szolgál, amely létrehozásakor két alapmodelli szabályt vár paraméterül. Egy `PriorityRule` típusú szabály alkalmazhatóságának vizsgálata során először a nagyobb prioritással rendelkezdő szabály alkalmazhatóságát kell vizsgálni, majd annak esetleges sikertelensége esetén lehet a kisebb prioritással rendelkező szabály alkalmazhatóságát elemezni.

3.3.4. Régiók



3.5. ábra. A membránrendszer régióit reprezentáló osztály

Egy régió azonosítására a saját egyedi azonosítóját (`id`) használja az alkalmazás, amely megegyezik a membránstruktúrában a hozzá tartozó `Node` objektum `uid` adattagjában tárolt értékkel. Ez az érték a rendszer számítása során nem változik, egyedül egy régió szülőjének címkéje változhat meg (alapmodell esetén feloldódás következtében), ezért ezt az értéket nem lehet statikusan letárolni. A feloldódás jelzésére az `is_dissolving` adattag hivatott, amelynek igaz értékre billenése esetén az evolúciós lépés végén a régió felbomlik. A régió emellett felveszi a `new_objects`

adattagot is, amely azon objektumok eltárolására szolgál, amelyek az éppen zajló evolúciós lépésben kerültek a régióba. Ez azért fontos, mert a maximális párhuzamosság elve szerinti szabályok kiválasztása egyidejűleg történik meg az ütem elején, tehát az egyik szabály alkalmazásának mellékhatása nem befolyásolhatja a kiválasztott szabályokat. Ezért amikor alkalmazásra kerül egy szabály, a jobb oldalán található objektumok elrejtésre kerülnek a további szabályok elől, majd az evolúciós lépésben a maximális szabálymultihalmaz után kerülnek a régió `objects` multihalmazába.

3.4. Forráskódok

Nulla sodales purus id mi consequat, eu venenatis odio pharetra. Cras a arcu quam. Suspendisse augue risus, pulvinar a turpis et, commodo aliquet turpis. Nulla aliquam scelerisque mi eget pharetra. Mauris sed posuere elit, ac lobortis metus. Proin lacinia sit amet diam sed auctor. Nam viverra orci id sapien sollicitudin, a aliquam lacus suscipit. Quisque ac tincidunt leo 3.2. és 3.3. forráskód:

```
#include <stdio>

int main()
{
  int c;
  std::cout << "Hello World!" << std::endl;

  std::cout << "Press any key to exit." << std::endl;
  std::cin >> c;

  return 0;
}
```

3.2. forráskód. Hello World in C++

```
using System;
namespace HelloWorld
{
  class Hello
  {
    static void Main()
    {
```

```
8         Console.WriteLine("Hello World!");
9
10        Console.WriteLine("Press any key to exit.");
11        Console.ReadKey();
12      }
13    }
14  }
```

3.3. forráskód. Hello World in C#

3.4.1. Algoritmusok

Az 1. algoritmus egy általános elágazás és korlátozás algoritmust (*Branch and Bound algorithm*) mutat be. A 3. lépésben egy megfelelő kiválasztási szabályt kell alkalmazni. Példa forrása: Acta Cybernetica (ez egy hiperlink).

1. algoritmus A general interval B&B algorithm

Funct IBB(S, f)

1: Set the working list $\mathcal{L}_W := \{S\}$ and the final list $\mathcal{L}_Q := \{\}$
2: **while** ($\mathcal{L}_W \neq \emptyset$) **do**
3: Select an interval X from $\mathcal{L}_W$ ▷ Selection rule
4: Compute $lbf(X)$ ▷ Bounding rule
5: **if** X cannot be eliminated **then** ▷ Elimination rule
6: Divide X into X^j, $j = 1, \ldots, p$, subintervals ▷ Division rule
7: **for** $j = 1, \ldots, p$ **do**
8: **if** X^j satisfies the termination criterion **then** ▷ Termination rule
9: Store X^j in $\mathcal{L}_W$
10: **else**
11: Store X^j in $\mathcal{L}_W$
12: **end if**
13: **end for**
14: **end if**
15: **end while**
16: **return** $\mathcal{L}_Q$

4. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusjegyzék

Forráskódjegyzék