Szakdolgozat

Bálint Attila

2012

Pannon Egyetem

Matematika Tanszék

Programtervező informatikus BSc

Szakdolgozat

A logika tanításának számítógépes támogatása

Készítette:

Bálint Attila

Témavezető:

Dr. Szalkai István

2012.

(feladatkiírás)

**Nyilatkozat**

Alulírott Bálint Attila diplomázó hallgató, kijelentem, hogy „A logika tanításának számítógépes támogatása” című szakdolgozatot a Pannon Egyetem Matematika Tanszékén készítettem programtervező informatikus BSc diploma (Computer Program Designer, BSc) megszerzése érdekében.

Kijelentem, hogy a szakdolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2012. április 26.

aláírás

Alulírott Dr. Szalkai István témavezető kijelentem, hogy „A logika tanításának számítógépes támogatása” című szakdolgozatot Bálint Attila a Pannon Egyetem Matematika Tanszékén készítette programtervező informatikus BSc diploma (Computer Program Designer, BSc) megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2012. április 26.

aláírás

**Köszönetnyilvánítás**

Legnagyobb köszönettel tartozom témavezetőmnek, Dr. Szalkai Istvánnak, aki hatalmas segítséget nyújtott dolgozatom elkészítésében, rendkívül hasznos tanácsokkal látva el hétről hétre. Köszönöm családomnak a tanulmányaim során nyújtott szerető támogatást és türelmet, barátnőmnek pedig a kitartást. Végül, de nem utolsó sorban köszönet illet mindenkit, aki szakdolgozatom elkészítésének ideje alatt személyes tanácsaival segítette munkámat.

**Tartalmi összefoglaló**

A szakdolgozat témája a logikai algebra oktatásának támogatása, a tanulási folyamat megkönnyítése számítógépes program segítségével. Dolgozatomban röviden áttekintem a Boole-algebrák különböző megjelenési formáit, ezek közül kiemelten foglalkozom a halmazalgebra területével. Többek között e terület segítségét felhasználva mutatom be egy segédprogramon keresztül a logikai kifejezések megoldásának menetét. A szoftver használata során lehetőség nyílik a logika alapjainak elsajátítására, de a fő hangsúlyt a komplexebb feladatok megoldásának kiértékelésére fordítottam. Számos funkció támogatja a megoldások kielemzését, változatos ábrázolásmódokat biztosítva. A szoftver tervezése és megvalósítása teljes egészében sajátkezű, önálló munkám eredménye, melyet számos teszt jellegű számítás elvégzése követett. A dolgozatban ezek eredményeivel, valamint képernyőképekkel illusztrálva mutatom be a programot.

**Kulcsszavak:** logika, Boole-algebra, halmazalgebra, konjunktív normálforma, diszjunktív normálforma, igazságtábla, Venn-diagram.

**Abstract**

The topic of the thesis is supporting the education of logic algebra, helping the learning process with a computer program. In this thesis I give a brief review of representations of different types of Boolean algebras, emphasizing the use of algebra of sets. I introduce the program and show the procedure of solving a logical expression with the help of this type of algebra, among others. In the course of using the software, the student is able to learn the basics of mathematical logic, but the main point is the evaluation of more complex expressions. Numerous functions support the analysis of results, ensuring a wide variety of data representation. The whole design and implementation processes of the software are the results of my own, individual work, which were followed by a number of test evaluations. I introduce the program with the results of these tests, also with the help of several screenshots in this thesis.

**Keywords:** logic, Boolean algebra, algebra of sets, conjunctive normal form, disjunctive normal form, truth table, Venn diagram.

**Tartalomjegyzék**

1. Irodalmi áttekintés 8

2. Problémafelvetés 10

3. Logika a matematikában 11

3.1. A logika tudománya 11

3.2. Boole-algebra 11

3.2.1. Boole-algebrák kapcsolatai 12

3.2.2. Absztrakt algebra 13

3.3. A Boole-algebra reprezentációi 14

3.3.1. Logikai műveletek 14

3.3.2. Halmazalgebra 15

3.3.3. Eseményalgebra 16

3.3.4. További megjelenések 18

3.3.5. Összegzés 19

3.4. Logikai kifejezések ábrázolása 19

3.4.1. A vizsgált ábrázolási módok 19

3.4.2. Ábrázolás igazságtáblán 20

3.4.3. Ábrázolás Venn-diagramon 21

4. A megvalósítás folyamata 22

4.1. Felhasználói igények felmérése 22

4.2. A szoftver elkészítése 23

4.2.1. A kiértékelés megvalósítása 23

4.2.2. Igazságtáblás ábrázolás megvalósítása 25

4.2.3. Venn-diagramos ábrázolás megvalósítása 26

4.2.4. További eredmények megjelenítése 28

4.2.5. Egyéb funkciók 28

5. Rendszerterv és implementáció 30

5.1. Előzetes követelmények megfogalmazása 30

5.2. Használati esetek, a szoftver funkciói 31

5.3. Az osztályokról 35

5.4. Eljárások implementálása, függvények 37

5.4.1. Kiértékelési mód váltás 37

5.4.2. Változók adatainak frissítése 38

5.4.3. Kifejezés kiértékelése 40

5.4.4. Igazságtábla felírása 41

6. Felhasználói ismertető 43

6.1. Előkészítés, környezet 43

6.1.1. Futtatás 44

6.2. A program használata 44

6.2.1. Indítás után 44

6.2.2. Gyors mód 46

6.2.3. Részletes mód 46

6.2.4. Jegyzetek 50

6.2.5. Menük 51

7. Futtatási tapasztalatok 53

7.1. Elvégzett tesztek 53

7.1.1. Teszt 1 54

7.1.2. Teszt 2 54

7.1.3. Teszt 3 55

7.2. Levont következtetések 55

7.3. Kiegészítés 57

*Irodalomjegyzék* 58

# 1. Irodalmi áttekintés

A matematikai logika irodalma igen jelentős, hiszen egy régóta ismert, sokat kutatott témáról van szó. Ebben a fejezetben érintőlegesen áttekintem ezen terület jelentősebb eredményeit, a legfontosabb művekkel együtt.

Elsőként a 19. század közepén George Boole [2], valamint Augustus De Morgan [4] formalizált bizonyos logikai fogalmakat az algebra segítségével, megteremtve ezzel a matematikai logika alapjait. Boole-algebráról 1860 óta beszélhetünk, Charles Peirce cikkeinek köszönhetően [12], aki megalkotta a relációk és kvantorok logikai rendszerét. Azóta számos tanulmány született a témával kapcsolatban. Megemlítendő Frege munkája is [5], amely mű megalapozta az újkori logikát. Schröder 1890 és 1905 között 3 részletben publikálta írását [14], melyben összegezte és továbbfejlesztette Boole, De Morgan, és Peirce munkáit.

A 20. század elejének kutatásai főként a halmazelmélet és a formális logika területeire irányultak. Russel nevéhez fűződik a naiv halmazelmélet ellentmondásosságának felfedezése [13], melyhez kapcsolódóan David Hilbert 1900-ban összeállított egy 23 problémából álló listát. A Boole-algebra első axiomatikus algebrai struktúraként történő tárgyalása 1904-ben, Edward Vermilye Huntington dolgozatával kezdődött meg. Leopold Löwenheim és Thoralf Skolem mondták ki a Löwenheim–Skolem tételt, miszerint az elsőrendű logika nem képes értelmezni a végtelen struktúrák számosságát. Skolem megmutatta [15], hogy ez a tétel igaz a halmazelmélet elsőrendű formalizálására, és minden olyan formalizálásra, ami megszámlálható modellel rendelkezik. Gödel 1929-ben doktori disszertációjában bebizonyította teljességi tételét, majd 1931-ben publikálta első nemteljességi tételét [6].

A Boole-algebra komoly matematikává Stone [16], valamint Birkhoff [1] munkáinak köszönhetően vált. Az 1960-as években Paul Cohen új eredményeket talált a matematikai logika és az axiomatikus halmazelmélet területén a Boole-algebra ágainak használatával, melyet 1966-os művében [3] ismertetett.

A logika alapjainak elsajátítása, összefüggéseinek megértése szempontjából nagyon hasznosnak találtam Szalkai írását [17], mely átfogó képet ad a matematikai logikáról.

A logikai programozás fejlődésében kulcsszerepet vállalt Alain Colmerauer 1972-ben kifejlesztett programozási nyelve, a „Prolog”, melyhez nem sokkal később Szeredi Péter is kifejlesztett egy interpretert Magyarországon. A nyelvet jellemzően mesterséges intelligencia alkalmazások megvalósítására, valamint a természetes nyelvészet eszközeként is használják. Közelebbi megismerésében a Wikipédia „Prolog” oldala [10] segített. A témához kapcsolódóan remek írásnak találtam Márkus és Márkusz tanulmányát [11], melyben hétköznapi, humoros problémákon keresztül vezetik be az olvasót a logika rejtelmeibe, valamint e problémák megoldását mutatják be „Prolog” és „Pascal” nyelvű programok segítségével.

A matematikai logika történetének áttekintésében, valamint a legfontosabb események és publikációk összegyűjtésében a Wikipédia matematikai logika történetéről szóló oldala [9] volt segítségemre.

# 2. Problémafelvetés

Az évek folyamán a logika jelentősége kulcsfontosságúvá vált mind a matematika, mind a számítástechnika tudományában. Gyakori alkalmazása mellett azonban úgy gondolom, megfelelő oktatása sok esetben háttérbe szorul, mely elsősorban az idő hiányának tudható be. Az oktatóknak nincs elég idejük megfelelő mennyiségű példafeladat megoldására, a diákok pedig a legritkább esetben állnak neki önállóan gyakorolni, így logikai tudásuk gyakran hiányos.

Meglátásom szerint a számítógépes programok nyújtotta lehetőségek kiaknázása leszűkül a feladatok megoldására, ami ily módon természetesen egyszerűbb, viszont nem segít a diáknak a probléma megértésében, az önálló gondolkodásban. A dolgozat célja ezen lehetőségek kibővítése az oktatás területére, melyek jelentősen megkönnyítenék a diákok számára a felkészülést. Ilyen, logikai kifejezéseket kiértékelő programot nehéz találni, de ha sikerrel is járunk, a megoldáshoz vezető lépések bemutatása hiányzik, így a felhasználó hiába kapja meg a helyes eredményt, egyáltalán nem biztos, hogy azt értelmezni is tudja. Dolgozatomban erre kívánok megoldást adni egy olyan szoftver formájában, amely nem csak egy adott logikai kifejezés kiértékelésére képes - mellyel a diák ellenőrizheti saját megoldását -, hanem emellett szemlélteti a felhasználóval a megoldáshoz vezető kulcslépéseket, megérteti vele a megoldás folyamatát. Változatos módokon képes ábrázolni az eredményt a Boole-algebra különböző reprezentációi segítségével. Ezen ábrázolásmódoknak köszönhetően a diák nagyobb eséllyel érti meg a logikai algebra összefüggéseit, könnyebben lesz képes értelmezni a megoldandó logikai feladatokat, valamint képessé válik ezek önálló megoldására. A célkorosztályt elsősorban a középiskolás, valamint a felsőfokú tanulmányaikat kezdő diákok képezik.

# 3. Logika a matematikában

## 3.1. A logika tudománya

A logika szó a görög „logos” szóból származik, melynek jelentése gondolat, beszéd, érv. Első, filozófiai megközelítése óta számos tudományban megjelent, legjellemzőbb ezek közül a nyelvészet és a matematika. A logikát a hétköznapokban a legkülönbözőbb területeken, széles körben alkalmazzuk érvelések, következtetések, feltételek formájában.

Formális megfogalmazásában a logika az érvényes következtetések és bizonyítások, valamint a hozzájuk kapcsolódó, legkülönfélébb területekről (főleg filozófia, matematika, nyelvészet) származó kérdések tudománya. Alapfeladata olyan formai kritériumok feltárása, amelyek szerint adott igaz, vagy igaznak feltételezett *p*1*, p*2*, ..., pn* állítások (premisszák) esetén helyesen következtethetünk egy *q* kijelentés (konklúzió) igazságára.

Dolgozatomban a matematikai logika egy szűkebb területével, az ítéletkalkulussal (ítéletlogika) foglalkozom. Az ítéletkalkulus az elsőrendű logika azon speciális este, amikor csak nulladrendű predikátumszimbólumok vannak, és azokból megszámlálhatóan végtelen sok. Ebben az esetben az elsőrendű kvantorok feleslegessé válnak, ezért elhagyjuk őket. Az ítéletkalkulus vagy ítéletlogika fő problémája a kijelentések között értelmezhető műveletek, a logikai műveletek, illetve a különféle levezetések tanulmányozása.

A fent említett kvantorokkal és a hozzá kapcsolódó állításkalkulussal a matematika egy szűkebb ága foglalkozik, ezekről bővebben Horváth, Joó és Szalkai írásában olvashatunk [7].

## 3.2. Boole-algebra

A matematikában, azon belül is az algebrában a Boole-algebra az a kétműveletes algebrai struktúra, amely a halmazműveletek, a logikai műveletek és az eseményalgebra műveleteinek közös tulajdonságaival rendelkezik. Matematikai oldalról vizsgálva a Boole-algebra olyan legalább kételemű, egység- és zéruselemes háló, mely disztributív és komplementumos.

### 3.2.1. Boole-algebrák kapcsolatai

Boole mutatott rá először, hogy a logikai algebra számos matematikai területen alkalmazható. Ezt követően több matematikus is vizsgálta ezen területeket, és a köztük fennálló algebrai jellegű kapcsolatot. Ilyen szoros összefüggések mutatkoznak például az alábbi területeken [8]:

* Logikai műveletek: az igazságértékek {0, 1} halmaza, a logikai „VAGY”, valamint a logikai „ÉS” műveletével. Az *a* elemhez tartozó ¬*a* kifejezés pedig az elem negációja, jelentése „NEM” *a*.
* Halmazműveletek: egy tetszőleges *H* halmaz hatványhalmaza, a *H* részhalmazai közötti unió és metszet tulajdonsággal. Az *A* részhalmaz komplementere a *H* azon elemei, melyek nincsenek benne *A*-ban.
* Eseményalgebra: a valószínűség-elmélet egy *K* eseménytere, az események közötti összeg és szorzat műveletével. Ebben az esetben *A* komplementere az az esemény, hogy az *A* esemény nem következik be.
* Egy *n* négyzetmentes, természetes számra *n* pozitív osztói disztributív hálót adnak, ha a rendezés az oszthatóság, a metszet a legnagyobb közös osztó, az egyesítés a legkisebb közös többszörös képzése. Ebben a hálóban a legkisebb elem az 1, a legnagyobb elem pedig *n*.

Dolgozatomban ezen területek közül az első hárommal foglalkozom részletesebben, ezek összefüggéseit használom fel és alkalmazom programomban. A fent említett területek mellett annak fényében, hogy az igaz-hamis értékek bináris számokkal vagy logikai áramkörök feszültségszintjeivel is azonosíthatók, megállapíthatjuk, hogy a párhuzam ezekre is fennáll.

Jellemző Boole-algebrák tehát például: halmazalgebra, eseményalgebra, ítéletek algebrája, osztók algebrája, kétállapotú kapcsolók algebrája. Jól látható, hogy a logikai algebra elmélete rengeteg gyakorlati alkalmazással bír a számítástechnika és a matematikai logika területén egyaránt, hiszen egy jól kiforrott, remekül általánosítható dologról van szó.

### 3.2.2. Absztrakt algebra

A fent megismertek alapján elmondható tehát, hogy a Boole-algebra absztrakt algebra, azaz műveleteinek definiálásához szükség van egy konkrét megjelenésére. Ezen megjelenésekre a 3.3. alfejezetben térek ki részletesebben. Általánosságban a Boole-algebra leírható azonban a következő struktúrával:

***B =*** (***H***, ****, ****,, **О**, **І**),

ahol ***H*** ≠ Ø az alaphalmaz,

**** és **** kétváltozós műveletek,

 egyváltozós művelet,

**О**, **І**∈ ***H*** speciális elemek.

A műveletek között azonosságokat követelünk meg, melyek *A*, *B*, *C* ∈ ***H*** esetén a következők:

1. *A* ****(*B* **** *C*)=(*A* **** *B*)**** *C* asszociativitás

2. *A* ****(*B* **** *C*)=(*A* **** *B*)**** *C*

3. *A* **** *B* = *B* **** *A* kommutativitás

4. *A* **** *B* = *B* **** *A*

5. *A* ****(*A* **** *B*)= *A*  abszorptivitás

6. *A* ****(*A* **** *B*)= *A*

7. *A* ****(*B* **** *C*)=(*A* **** *B*)****(*A* **** *C*) disztributivitás

** 8. *A* ****(*B* **** *C*)=(*A* **** *B*)****(*A* **** *C*)

** 9. *A* **** *A* = **І** komplementerek

** 10. *A* **** *A* =**О**

** 11. **О**= **І**

12. **І**=**О**

** 13.(*A*)= *A* kettős tagadás

14. *A* **** *A* = *A* idempotencia

15. *A* **** *A* = *A*

16. *A* ******О**= *A* korlátosság

17. *A* ******І** = *A*

18. *A* ******І** = **І**

** 19. *A* ******О**=**О**

** 20.(*A* **** *B*)= *A* **** *B* De Morgan-azonosságok

21. (*A* **** *B*)= *A* **** *B*

Bármely algebra esetén, amennyiben a fenti azonosságok teljesülnek, Boole-algebráról beszélhetünk. Másik oldalról megközelítve a dolgot megállapítható tehát, hogy ezen tulajdonságok - ahogy azt a következő alfejezetben részletesen ismertetem - jellemzők a Boole-algebra összes reprezentációjára.

## 3.3. A Boole-algebra reprezentációi

### 3.3.1. Logikai műveletek

A Boole-algebra ezen reprezentációjában az alaphalmazt (*H*) az igazságértékek {0, 1} halmaza jelenti, a három logikai alapművelet, valamint a hozzájuk tartozó operátorok a következők:

** Diszjunkció műveletének operátora („VAGY”művelet).

** Konjunkció műveletének operátora („ÉS”művelet).

¬ Negáció műveletének operátora („NEM”művelet).

Legkisebb elemnek a logikai hamis (0) érték, legnagyobb elemnek pedig a logikai igaz (1) érték felel meg. A fenti 3 művelet segítségével képesek vagyunk bármely logikai kifejezés felírására.

A 3.2.2. pont azonosságai - az eddig elmondottak értelmében - megtalálhatók tehát a logikai műveleteknél is. Ezek felírásához mindössze a logikai műveletek jeleinek behelyettesítésére és a megfelelő elemek kicserélésére van szükség.

Néhány példa erre:

1. *A* (*B*  *C*)=(*A*  *B*) *C* asszociativitás

2. *A * (*B  C*)=(*A  B*) * C*

9. *A * (¬*A*)=1 komplementerek

10. *A * (¬*A*)=0

16. *A * 0= *A* korlátosság

17. *A * 1= *A*

20.¬(*A  B*)=(¬*A*) ** (¬*B*) De Morgan-azonosságok

21. ¬(*A  B*)=(¬*A*) ** (¬*B*)

Logikai műveletek esetén a fent említettek mellett definiálhatunk további, összetett műveleteket is, melyek - a korábban elmondottak alapján - felírhatók a 3 alapművelet segítségével. Ilyen jellemző műveletek például:

 Implikáció operátora: olyan kétváltozós műveletet jelöl, mely *A*  *B* esetén csak abban az esetben hamis, ha *A* logikai értéke igaz, *B* logikai értéke pedig hamis. Alapműveletekkel kifejezve: *A  B* =¬*A  B*.

 Ekvivalencia operátora: olyan kétváltozós műveletet jelöl, mely *A*  *B* esetén *A* és *B* logikai értékének egyezése esetén lesz igaz, tehát ha *A* és *B* logikai értéke egyszerre igaz, vagy egyszerre hamis. Alapműveletekkel kifejezve: *A  B* =(*A*  *B*) ** (¬*A*  ¬*B*),

vagy másképp: *A  B* =(¬*A  B*) ** (¬*B * *A*).

### 3.3.2. Halmazalgebra

Egy másik lehetséges megjelenési forma a halmazalgebra, melynek tényét Stone reprezentációs tételéből - mely kimondja, hogy bármely Boole-algebra izomorf egy halmazalgebrával - ismerhetjük. Stone megközelítésében a halmazalgebra áll a központban, nézete szerint a többi Boole-algebra tulajdonságai ennek segítségével írhatók le. A halmazalgebra esetében az alaphalmaz egy adott *H* halmaz P(*H*)hatványhalmaza lesz. A halmazalgebra műveletei a következők:

∪ Unió (egyesítés) műveletének operátora.

 Metszet (közös rész) műveletének operátora.

ˉ Komplementerképzés (alaphalmazból való halmazkivonás) műveletének operátora.

Ebben az esetben a legnagyobb elem az alaphalmaz (P(*H*)), a legkisebb elem pedig az üres halmaz (Ø).

A korábban bemutatott műveletkiosztások ismeretében a 3.2.2. pontban tárgyalt összes azonosság kiterjeszthető a halmazalgebrára, a megfelelő elemek felhasználásával.

Ezek közül néhány:

1. *A* U(*B* U *C*)=(*A* U *B*)U *C* asszociativitás

2. *A* (*B*  *C*)=(*A*  *B*)  *C*

** 9. *A* U *A* = *H* komplementerek

10. *A*  *A* =Ø

16. *A* UØ= *A* korlátosság

** 17. *A*  *H* = *A*

** 20. *A* U *B* = *A*  *B* De Morgan-azonosságok

21. *A*  *B* = *A* U *B*

### 3.3.3. Eseményalgebra

Eseményalgebrának a valószínűségszámításhoz szükséges struktúrát nevezzük. Ebben a megközelítésben a kísérletet egy halmazzal azonosítjuk, a kísérlet kimeneteleinek *K* halmazával, melyet eseménytérnek is hívhatunk. Mindent, amiről a kísérlet elvégzése után eldönthető, hogy bekövetkezett-e vagy sem, eseménynek nevezünk. A kísérlet egy megismétlése mindig egy kimenetelt ad, melynek ismeretében egyértelműen eldönthető, hogy egy adott *A* esemény, mint a kimenetelről szóló állítás, teljesül-e vagy sem. Azon kimenetelek, melyek bekövetkezése az *A* állítást igazzá teszi, a *K* egy részhalmazát alkotják. A kimenetelek is felfoghatóak eseményeknek, hiszen a *k*∈*K* kimenetelhez egyértelműen tartozik egy {*k*}⊆*K* egyelemű esemény, azaz elemi esemény. Az eseményeken értelmezett műveletek a következők:

**** Események összeadása: eredményét azon elemi események alkotják, melyek vagy az egyik, vagy a másik eseménynek elemei.

 Események szorzása: eredményét azon elemi események alkotják, melyek mindkét eseménynek elemei.

ˉ Esemény komplementere: eredménye minden olyan elemi esemény, mely nem eleme az eseménynek.

Ezen reprezentációban a legkisebb elem a 0, mint lehetetlen esemény, a legnagyobb elem pedig az 1, mint biztos esemény, amely a kísérlet során mindig bekövetkezik. Komplementer eseményekről abban az esetben beszélhetünk, ha két egymást kizáró esemény együttesen épp a biztos eseményt adja.

A fentiek ismeretében a 3.2.2. pont azonosságai alkalmazhatók az eseményalgebra esetében is, az események összeadása, események szorzása, valamint az esemény komplementere műveletek alkalmazásával.

Ezekre néhány példa:

1. *A* **** (*B* **** *C*)=(*A* **** *B*) **** *C* asszociativitás

2. *A* (*B*  *C*)=(*A*  *B*)  *C*

** 9. *A* **** *A* =1 komplementerek

10. *A*  *A* =0

16. *A* ****0= *A* korlátosság

17. *A* 1= *A*

**** 20. *A* **** *B* = *A*  *B* De Morgan-azonosságok

21. *A*  *B* = *A* **** *B*

Leegyszerűsítve tehát, valamint ahogy azt később a program használata során látni fogjuk, egy *n* változós kifejezés esetén úgy kapjuk meg az esemény bekövetkezési valószínűségét, hogy az összes lehetséges esetből (összes bemeneti variáció, azaz 2*n* db lehetőség) csak azokat vesszük figyelembe, amelyek igazzá teszik a kifejezést. Ezen bemeneti kombinációk megegyeznek a továbbiakban (3.4.2. pont) ismertetett igazságtábla soraival, ami azt jelenti, hogy az igazságtábla felírása után a bekövetkezési valószínűség egyszerűen számolható az igazra kiértékelt sorok és az összes sor számából.

Venn-diagramon való ábrázolás esetében (3.4.3. pont) hasonló módon számolhatunk, itt a színezett halmazterületek számát vetjük össze az összes halmazterülettel. Ezek alapján jól látható a szoros összefüggés adott kifejezéshez tartozó igazságtábla, valamint a hozzá tartozó Venn-diagram között. Ezen kapcsolattal a 3.4. alfejezetben foglalkozom részletesebben.

### 3.3.4. További megjelenések

A 3.2.1. pontban megemlített Boole-algebrák között szerepelt egy további terület is, méghozzá az osztók algebrája. Ehhez a reprezentációhoz szükségünk van egy *n* négyzetmentes, természetes számra. A *Dn* alaphalmazt *n* osztói alkotják, műveletek a legnagyobb közös osztó (lnko) és a legkisebb közös többszörös (lkkt) képzése. A legkisebb elem az 1, a legnagyobb elem pedig *n*. Egy *x* természetes szám komplementerén -et értjük.

A fent bemutatottak ismeretében a 3.2.2. pontban tárgyalt azonosságok kiterjeszthetők a Boole-algebra ezen reprezentációjára is, a megfelelő műveletek és elemek kicserélésével. Ezek közül néhány:

1. lkkt(*A,* lkkt(*B*, *C*))=lkkt(lkkt(*A*, *B*), *C*) asszociativitás

2. lnko(*A*lnko(*B* *C*))= lnko(lnko(*A*, *B*), *C*)

9. lkkt(*A*,  )= *n* komplementerek

10. lnko(*A*,)=1

16. lkkt(*A*,1)= *A* korlátosság

17. lnko(*A*, *n*)= *A*

20. = De Morgan-azonosságok

21. =

### 3.3.5. Összegzés

A fentiekből látható, hogy a Boole-algebrának számos megjelenési formája létezik a matematika különböző területein. Ezen reprezentációk mindegyikére alkalmazható a 3.2.2. pontban felírt általános struktúra. Elmondható tehát, hogy e struktúra elemeinek ismeretében képesek vagyunk a Boole-algebra bármely megjelenési formájának értelmezésére, nincs szükség új definíciók elsajátítására, csupán az alaphalmazt, a legkisebb és legnagyobb elemet, valamint az ezeken értelmezett műveleteket kell meghatároznunk.

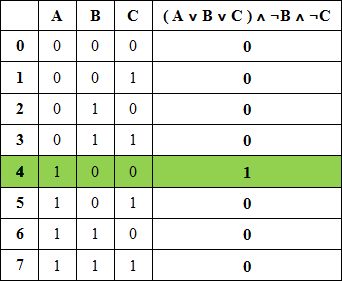
## 3.4. Logikai kifejezések ábrázolása

### 3.4.1. A vizsgált ábrázolási módok

Vegyük például a következő kifejezést:( *A  B  C* ) ** ¬*B *  ¬*C*. A korábban tárgyaltak szerint megállapíthatjuk, hogy ezen kifejezés megoldását többféleképpen is ábrázolhatjuk. Felírhatjuk például a kifejezésben szereplő változók logikai értékeinek kombinációjaként, minden esetet külön kiértékelve. Ezen logikai értékeket az egyes esetekhez rendelve ábrázolhatjuk a megoldást igazságtáblával (1. ábra), valamint a 3.3.2. pontban megfogalmazottak szerint ábrázolhatjuk Venn-diagramon is (2. ábra).

### 3.4.2. Ábrázolás igazságtáblán

Igazságtáblán ábrázolva a fenti ( *A  B  C* ) ** ¬*B*  ¬*C* kifejezést a következő táblázatot kapjuk:



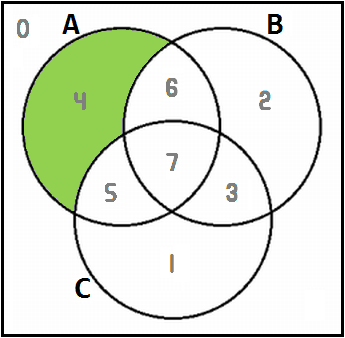
1. ábra. Megoldás ábrázolása igazságtáblán

A táblából könnyen kiolvasható, hogy a kifejezés abban az esetben lesz csak igaz, ha *A* igaz, *B* és *C* pedig hamis. A változók előtti első oszlop a vizsgálandó eset sorszáma (0-val kezdődően), amely minden táblázat esetén 2*n*db sort jelent, ahol *n* a változók száma. Ezen értékkel egyszerűen hivatkozhatunk az adott esetre. Ha más ábrázolási módot használunk, célszerű az azonos esetekhez azonos sorszámokat rendelnünk, így könnyen tehetünk utalásokat a kifejezés különböző reprezentációi között. A táblázat elkészítésénél fontos, hogy a változók értékeiből alkotott bináris szám az igazságtáblában növekvő sorrendben szerepeljen, megkönnyítve a könnyebb olvashatóságot. E szám decimális értéke lesz az adott sor sorszáma.

Mivel az igazságtábla mérete a változók számának növelésével exponenciálisan növekszik (2*n*) és ez a méret nem csökkenthető, kellően nagy *n* esetén ez jelentősen lelassíthatja a program futását. Ehhez kapcsolódóan számos tesztet elvégeztem, melyek eredményeit a 7. fejezetben ismertetem.

### 3.4.3. Ábrázolás Venn-diagramon

Venn-diagramon ábrázolva ( *A  B  C* ) ** ¬*B*  ¬*C* kifejezést a következő ábrát kapjuk:

****

2. ábra. Megoldás ábrázolása Venn-diagramon

Elmondható, hogy bármely igazságtáblához készíthető Venn-diagram, ebben az esetben pedig az igazságtábla minden egyes sorához a diagramon pontosan egy halmazterület fog tartozni. Ezen ábrázolásnál a halmazok egyes területeit azonos számmal jelölöm, mint az igazságtáblában. Ez azt jelenti, hogy például ( *A  B  C* ) ** ¬*B*  ¬*C* esetén az 1. ábrán látható ábrázolás szerint az 4-essel jelölt sor változói (*A*=1, *B*=0, *C*=0) teszik igazzá a kifejezést. A Venn-diagram esetében jól látható, hogy az a terület, mely része *A*-nak, *B*-nek és *C*-nek viszont nem, ugyancsak az 4-es számot kapta. Az előző pontban is említett sorrendiség betartása esetén elmondható, hogy bármely hasonló szabályok szerint elkészített Venn-diagram esetében a 0-ás számú terület azt az esetet jelöli, mikor az összes változó logikai értéke hamis, (2*n*-1)-esszámú terület pedig az összes halmaz metszete lesz, ahol minden változó logikailag igaz.

# 4. A megvalósítás folyamata

Célom egy olyan szoftver kifejlesztése, mely az előző fejezetben tárgyalt területek összefüggéseit a lehető legszélesebb körben, sok eredménnyel, látványos módon képes ábrázolni, ezáltal nagymértékben hozzájárul a felhasználó logikai tudásának fejlesztéséhez. Nagy hangsúlyt fektettem a különböző ábrázolási módok közti kapcsolatok szemléltetésére, ezzel is megkönnyítve az átláthatóságot, értelmezhetőséget.

## 4.1. Felhasználói igények felmérése

A kitűzött célok gyakorlati megvalósítása előtt nélkülözhetetlen volt a probléma mélyebbre ható, felhasználói szemszögből való tanulmányozása, melynek során a következő fő szempontokat vettem figyelembe:

1. A szoftver felülete legyen átlátható, könnyen kezelhető.
2. Az átláthatóságot szem előtt tartva jelenítse meg a lehető legtöbb szükséges információt, amely segítheti a feladat megértését.
3. Legyen képes az eredmények széleskörű ábrázolására, többek között diagramok, táblázatok formájában.
4. A témában kevésbé jártas felhasználók számára nyújtson megfelelő mennyiségű segédanyagot és támogatást (súgó).
5. Az éppen szükségtelennek vélt információt legyen lehetőség elrejteni, a felületet személyre szabni.
6. A felhasználó képes legyen munkáját elmenteni, később újra betölteni.
7. Objektum-orientált programozás támogatása a későbbiekben felmerülő igények miatt, bővíthetőség, karbantartás megkönnyítéséhez.

Ezen igények alapján világossá válik, hogy döntő fontosságú a szoftver felhasználói felülete, a fejlesztéshez ezért C# nyelvet, és Visual Studio 2010 fejlesztői környezetet használok, mely nagy segítséget nyújt a felület megtervezéséhez, számtalan beépített komponense felhasználásával pedig az adatok széleskörű reprezentációját is támogatja.

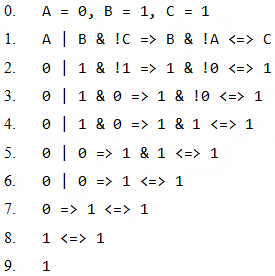
## 4.2. A szoftver elkészítése

Mivel a cél egy sokoldalú segédprogram fejlesztése, melynek lényege, hogy megkönnyítse a diákok számára a matematikai logika megértését, összefüggéseinek elsajátítását, egy olyan program létrehozása mellett döntöttem, melyet teljes mértékben saját magam által írt eljárásokból építek fel, azaz nem használok semmilyen előre megírt kódot. Ezzel elérem, hogy maximális mértékben felügyelni tudom a logikai kifejezések kiértékelésének folyamatát, azaz akár részeredmények felhasználására is lesz lehetőség. Erre egy átvett, kész komponens esetén nagy valószínűséggel nem lenne mód.

### 4.2.1. A kiértékelés megvalósítása

A szoftver készítésének kezdeti fázisában célom egy olyan alkalmazás elkészítése volt, mely egy előre definiált műveletkészletből (logikai műveletek), valamint változók neveiből álló kifejezésről képes eldönteni, hogy szintaktikailag helyes kifejezés-e, illetve szintaktikailag helyes kifejezés esetén, a műveletek precedenciáját alkalmazva a változók megadott logikai értéke alapján meghatározza a kifejezés eredményét.

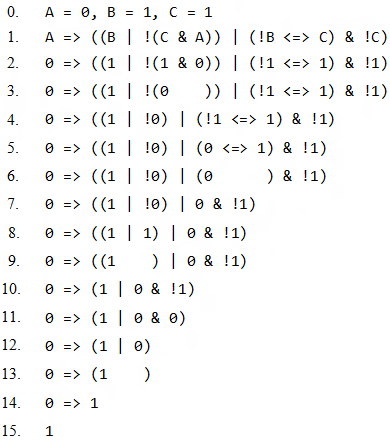
Ennek alapja egy a kifejezés kiértékeléséért felelős osztály, melynek kiértékelő metódusa egy karakterláncban megkapja a kiértékelendő kifejezést, melyet aztán a megfelelő precedenciavizsgálatokat követően, a soron következő művelettel megpróbál kiértékelni. Ez azt jelenti, hogy elsőként elemi negációkat keres a kifejezésben. Amennyiben van ilyen, a „¬0”karaktersorozatot „1”-re, a „¬1” karaktersorozatot pedig „0”-ra cseréli. Ha nem talált elemi negációt, vagy végzett a cserékkel, tovább folytatja a keresést, méghozzá elemi konjunkciót, majd ezt követően hasonló szabályok alapján elemi diszjunkciót, implikációt, ekvivalenciát keresve. Elemi művelet alatt az a művelet értendő, melyben a művelet operátora mellett kizárólag 0 és/vagy 1 szerepelnek. Ha az összes művelettel végzett, megkapjuk a kiértékelendő kifejezés eredményét, ami végül 0 vagy 1 lesz. Egy ilyen kiértékelési folyamat lépései láthatók a következő ábrán:



3. ábra. Az eljárás kiértékelésének részletes lépései zárójel nélküli kifejezés esetén, ahol „!” a negációt, „&” a konjunkciót, „|” a diszjunkciót, „=>” az implikációt, „<=>” pedig az ekvivalenciát jelenti

Következő lépésként a többszörösen összetett, többszörösen zárójelezett kifejezések helyes kiértékelésének megvalósítása volt a cél, amely jelentősen megnehezítette a munkát, hiszen a műveleti precedencia mellett a kifejezés helyes felbontására, valamint ezen partíciók kiértékelésének sorrendjére is figyelni kell.

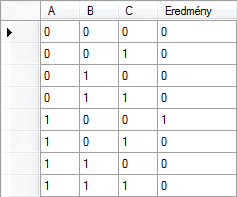
A kiértékelendő kifejezést a zárójelezés helyességének vizsgálatát követően karakterenként megvizsgáljuk aszerint, hogy az adott karakter hányadik zárójel szinten foglal helyet, majd ezt az értéket hozzárendeljük minden karakterhez. A zárójelen kívül szereplő karakterekhez rendelt érték ez esetben 0 lesz. Ezt követően az előbb kiszámolt szintek szerint felbontjuk a kiértékelendő kifejezést részkifejezésekre, melyeket egy listához adunk, majd ezen részkifejezések eredeti kifejezésben szereplő pozícióit is eltároljuk. Mivel a listához egymás után adjuk hozzá a részkifejezéseket, a lista végén szerepelnek majd a legbelső zárójelben található részkifejezések. Ezért a listában hátulról indulva hívjuk meg minden egyes részkifejezésre a korábban említett kiértékelő függvényt, majd a kapott eredményt visszahelyettesítjük a kiértékelt részkifejezés eredeti kifejezésben szereplő pozíciójára. Ezt folytatjuk mindaddig, míg a kifejezésben található összes zárójelet el nem tüntettük. Ha ez sikerült, a kifejezés egyetlen további függvényhívással kiértékelhető, a 3. ábrához hasonló módon. Egy teljes kiértékelési folyamat lépéseit szemlélteti a következő ábra:



4. ábra. Az eljárás kiértékelésének részletes lépései zárójelezett kifejezés esetén, ahol az egyes operátorok megegyeznek a 3. ábrán ismertetettekkel

### 4.2.2. Igazságtáblás ábrázolás megvalósítása

A sikeres kiértékelést követő lépés az igazságtábla kiszámolása volt, melyet a kiértékelő függvény többszöri hívásával oldottam meg, az előzőekben tárgyalt kiértékelés módszerét alkalmazva az igazságtábla minden sorára. Minden esetben az adott sorhoz tartozó változók logikai értékeit használjuk a kifejezésben szereplő változók értékeként. Mivel ez *n* változó esetén 2*n* db sort jelent, ezáltal 2*n* db függvényhívást végzünk. Az algoritmus elkészítése után az adatok táblázatban történő megjelenítésére legcélszerűbb megoldásként *DataGridView* komponenst használtam (5. ábra).

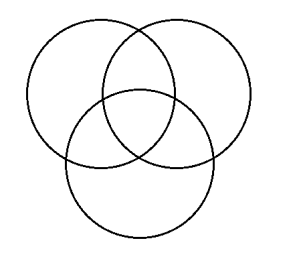


5. ábra. *DataGridView* control C#-ban

### 4.2.3. Venn-diagramos ábrázolás megvalósítása

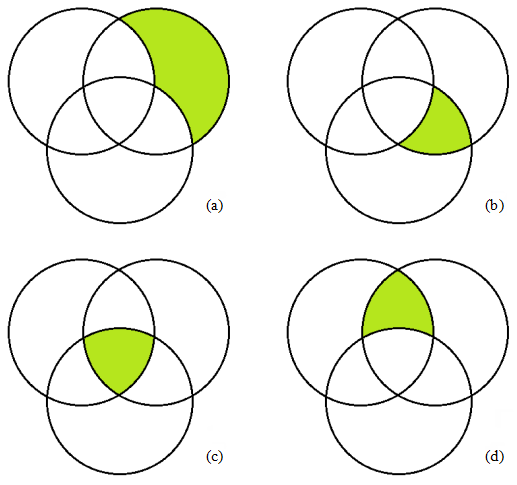
Az igazságtábla sikeres megvalósítása után a Venn-diagramon való ábrázolás elkészítésén dolgoztam. Mivel a 3.4. alfejezetben vizsgáltak alapján ismeretes, hogy egy *n* változós kifejezéshez tartozó igazságtábla minden sora megfeleltethető egy *n* halmazból álló Venn-diagram adott halmaz-területének, így ehhez az ábrázolásmódhoz felhasználtam a kifejezéshez korábban kiszámolt igazságtáblát. Ebből adódóan *n* db változó esetén szintén 2*n* db átlátszó képet használtam a Venn-diagram megjelenítésére, ahol minden kép egy-egy színezett halmazterületet ábrázol.

Első lépésként a változószámtól függően a program megjeleníti az üres halmazokat (6. ábra).



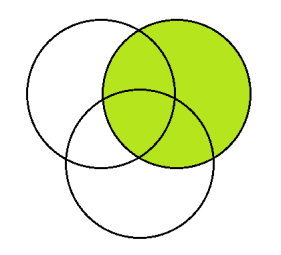
6. ábra. Üres halmazok 3 változó esetén

Ezt követően az igazságtábla eredményei alapján, az igaz logikai értékű sorokhoz tartozó halmazterületeket “kiszínezi”, méghozzá oly módon, hogy rárajzolja a szükséges halmazterületet tartalmazó képeket (7. ábra).



7. ábra. Négy különböző halmazterületet tartalmazó kép 3 változó esetén

Miután az eljárás végzett az igazságtábla összes sorával és a szükséges képeket kirajzolta a panelre, a végeredmény előállt, így már csak a panel megjelenítésére van szükség ahhoz, hogy a felhasználó számára is láthatóvá váljon a kész diagram. A fenti halmazterületek egymásra rajzolása után előállt diagramot a következő ábra szemlélteti:



8. ábra. A megjelenített végeredmény 7/a, 7/b, 7/c és 7/d kirajzolása után

Könnyen észrevehető tehát, hogy a teljes ábra kiszínezéséhez 3 változó esetén mind a 8 kép egymásra illesztésére lenne szükség ahhoz, hogy az összes halmazterületet lefedjük.

### 4.2.4. További eredmények megjelenítése

Az igazságtábla alapján előállítható a kifejezéshez tartozó diszjunktív, valamint konjunktív normálforma is, illetve eseményként értelmezve (3.3.3. pont) az adott esemény bekövetkezésének valószínűsége. Ezen eredmények egyaránt megjelenítésre kerülnek a programban, a kiértékelés lépéseit bemutató magyarázattal együtt. Ennek lépéseit a korábban (4.2.2. pontban) ismertetett felbontás részkifejezései alapján kapjuk meg.

### 4.2.5. Egyéb funkciók

További funkcióként lehetőség van a fent említettek (normálformák, valószínűség) kiszámolására a felhasználó által tetszőlegesen felírt igazságtáblából is. Ehhez a felhasználó számára lehetőséget kellett biztosítani igazságtábla létrehozására, melyben ő maga töltheti ki az eredmények oszlopát. Ennek megvalósítására ugyancsak az 5. ábrán bemutatott *DataGridView* komponenst használtam, melynek “Eredmények” oszlopát ez esetben értelemszerűen üresen hagytam. Jegyzetek létrehozásáról is gondoskodtam, melybe a felhasználó lejegyezheti a használat során szerzett tapasztalatait.

Látványosnak és hasznosnak találtam, hogy az igazságtábla tetszőleges sorait kijelölve a Venn-diagramon azonnal villogni kezdjenek a kijelölt sorokhoz tartozó halmazterületek, így egy időzítő segítségével ezt is megvalósítottam, ami nagyszerűen ábrázolja a logikai műveletek és a halmazműveletek között fennálló kapcsolatot.

Végül az összes eddig említett eredmény elmentését és – ahol indokolt volt - betöltését oldottam meg, melynek legösszetettebb része az Excel munkalapok helyes kezelésének megvalósítása volt. Emellett a program képes képek (PNG formátumban történő) előállítására is, a kifejezések elmentésére pedig egy saját fájlkiterjesztést hoztam létre (.xpr). A fájlműveletek többi része sima szöveges fájlként (.txt) írja és olvassa a fájlokat - többek között a normálformák vagy épp a jegyzetek mentése esetén -, hiszen fontosnak tartottam, hogy bárhol megnyithatóak legyenek ezen fájlok, akár egy egyszerű szövegszerkesztőből is.

Befejezésként a szoftvert részletes súgó funkcióval láttam el, amelyben igyekeztem a használat során szükségesnek ítélt legfontosabb információkat összefoglalni.

# 5. Rendszerterv és implementáció

## 5.1. Előzetes követelmények megfogalmazása

A 4.1. alfejezetben a szoftver igényeit felhasználói szemszögből vizsgáltam, ezúttal programozási oldalról közelítem meg a problémát. A program legjellemzőbb funkcionalitásait figyelembe véve a következő táblázatot állítottam össze, amely tartalmazza a kész szoftver esetén rendelkezésre álló kulcsfunkciókat, valamint a hozzájuk tartozó, sikeres megvalósításhoz szükséges alapvető követelményeket.

|  |  |
| --- | --- |
| **Tulajdonság** | **Követelmények** |
| Adatok beolvasása fájlból | Megfelelő fájltípusok felismerése  Hibás bemenet kezelése  Szükséges adatok azonosítása |
| Rendelkezésre álló adatokból különböző eredmények előállítása | Megfelelő számú speciális algoritmus egyes problémák kezelésére |
| Eredmények ábrázolása | Táblázatban való megjelenítés támogatása  Táblázatsorok kijelölésének detektálása  Diagramok megjelenítése  Diagramok színezése eltérő kritériumok alapján |
| Megjelenés személyre szabása | Kiértékelési módhoz kötött megjelenés  Lehetőség panelek elrejtésére |
| Adatok írása fájlba | Különböző fájltípusok (pl. Excel munkalapok) támogatása  Adatok exportálása formátumnak megfelelően |

A felsorolt tulajdonságok a szoftver fő funkcióit foglalják össze általánosan, programozástechnikai oldalról megközelítve. Az egyes problémák megoldásának megvalósítási folyamatára a 4.2., a konkrét implementációra pedig az 5.4. alfejezetben térek ki.

## 5.2. Használati esetek, a szoftver funkciói

A következőkben ismertetem a szoftver futása során előforduló használati eseteket, illetve rövid magyarázatot adok minden funkcióhoz.

|  |  |
| --- | --- |
| **Használati eset** | **1. Kiértékelési mód választása** |
| Leírás | Választás “Gyors”, illetve “Részletes” kiértékelés közül. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | Az aktuális kiértékelési mód nem felel meg a felhasználó további céljainak. |
| Előfeltételek | - |
| Sikeres lefutás | 1. A felhasználó rákattint a kívánt módra |
| Eredmény | A szoftver megjelenítette a kiválasztott módhoz tartozó nézetet. |
| Alternatív lefutás | - |

Kiértékelési mód választása: a felhasználó a megoldandó feladat jellege alapján eldönti, hogy milyen kiértékelési módot szeretné használni. Ha csupán egy kifejezés eredményére kíváncsi a „Gyors”, ha azonban részletes megoldást szeretne különböző ábrázolási módokkal a „Részletes” módot választja.

|  |  |
| --- | --- |
| **Használati eset** | **2. Változók meghatározása** |
| Leírás | A változók kívánt tulajdonságainak beállítása. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | Az aktuális változók száma és/vagy neve és/vagy értékei nem felelnek meg a felhasználó további céljainak. |
| Előfeltételek | - |
| Sikeres lefutás | 1. A felhasználó végrehajtja a kívánt változtatásokat |
| Eredmény | A szoftver frissítette a változók adatait. |
| Alternatív lefutás | 1a. A választott változónév már használatban van |

Változók meghatározása: a változók számának, nevének, valamint logikai értékének beállítása a felhasználó igényei szerint.

|  |  |
| --- | --- |
| **Használati eset** | **3. Kifejezés megadása** |
| Leírás | Logikai kifejezés megadása kiértékelés céljából. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | A felhasználó be szeretne vinni egy adott kifejezést a programba. |
| Előfeltételek | - |
| Sikeres lefutás | 1. A felhasználó megadja a kívánt kifejezést |
| Eredmény | A kifejezés megjelent a “Kifejezés” panelen. |
| Alternatív lefutás | - |

Kifejezés megadása: a kiértékelendő logikai kifejezés meghatározása. Másik lehetőségként a meghatározás történhet fájlból való betöltést követően is. Ebben az esetben csak „.xpr” kiterjesztésű fájlokat használhatunk.

|  |  |
| --- | --- |
| **Használati eset** | **4. Kifejezés kiértékelése** |
| Leírás | A “Kifejezés” panelen szereplő kifejezés kiértékelése. |
| Érintett aktor | Függvény |
| Kiváltó esemény | A felhasználó meg szeretné kapni a megadott kifejezés kiértékelésének eredményét. |
| Előfeltételek | A “Kifejezés” panel kifejezést tartalmazó mezője nem üres. |
| Sikeres lefutás | 1. A felhasználó rákattint a “Kiértékel” gombra  2. A program kiválasztja a megfelelő eljárásokat  3. Az eljárások lefutnak |
| Eredmény | A szoftver a választott módnak megfelelően megjelenítette a kiértékelés eredményeit. |
| Alternatív lefutás | 2a. A megadott kifejezés szintaktikailag helytelen |

Kifejezés kiértékelése: a megadott logikai kifejezés kiértékelése, valamint a kapott eredmények megjelenítése attól függően, hogy a felhasználó milyen kiértékelési módot választott. Ez a program legfontosabb, egyben legösszetettebb funkciója, melynek futási ideje a 3.4.2. pontban ismertetettek szerint exponenciálisan nő a változók számának növelésével. Kellően bonyolult, sokváltozós kifejezés esetén a kiértékelés ideje több másodpercet is igénybe vehet, a használt gép teljesítményétől függően. Erre bővebben a 7. fejezetben térek ki.

|  |  |
| --- | --- |
| **Használati eset** | **5. Igazságtábla létrehozása** |
| Leírás | Új igazságtábla létrehozása. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | A felhasználó kitölthető igazságtáblát szeretne létrehozni. |
| Előfeltételek | A kiválasztott mód “Részletes” |
| Sikeres lefutás | 1. A felhasználó rákattint az “Új igazságtábla” gombra  2. A program létrehozza az igazságtáblát |
| Eredmény | Az új igazságtábla megjelent az “Igazságtábla” panelen. |
| Alternatív lefutás | 1a. Létező igazságtábla esetén a program felajánlja a mentést |

Igazságtábla létrehozása: új igazságtábla készítése további feladatokban való felhasználás céljából. Ez két okból történhet: a felhasználó vagy normálformákat akar előállítani az általa megadott igazságtáblából, vagy egy a program által kiszámolt igazságtáblát szeretne önállóan megoldani, majd a megoldást leellenőrizni az eredeti igazságtábla alapján.

|  |  |
| --- | --- |
| **Használati eset** | **6. Jegyzet készítése** |
| Leírás | Jegyzet létrehozása szövegfájlként. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | A felhasználó megjegyzéseket szeretne írni egy feladattal kapcsolatban. |
| Előfeltételek | - |
| Sikeres lefutás | 1. A felhasználó rákattint az “Új jegyzet” gombra  2. A program létrehozza a jegyzetet |
| Eredmény | A jegyzet ablak megjelenik a képernyőn. |
| Alternatív lefutás | - |

Jegyzet készítése: a felhasználó létrehozhat tetszőleges jegyzeteket, melyek a későbbi feladatmegértésben nyújthatnak segítséget, tehát lehetőség van ezek mentésére, majd betöltésére is. Mivel a jegyzeteket a program sima szöveges fájlba menti, egyszerűen megnyithatók bármely egyszerű szövegszerkesztővel.

|  |  |
| --- | --- |
| **Használati eset** | **7. Adatok mentése** |
| Leírás | A megadott kifejezés, vagy a program által számolt eredmények fájlba mentése. |
| Érintett aktor | Felhasználó |
| Kiváltó esemény | A felhasználó el szeretné menteni a feladatot, vagy annak egy részét, hogy később újra felhasználhassa. |
| Előfeltételek | Mentés funkció aktív az adott panelen, azaz a panel tartalmaz adatot. |
| Sikeres lefutás | 1. A felhasználó rákattint a menteni kívánt részt tartalmazó panelen a “Mentés” ikonra  2. A felhasználó megadja a fájl nevét, amibe a program ment  3. A program elvégzi a mentést |
| Eredmény | A kívánt fájlok megjelennek a kiválasztott mappában. |
| Alternatív lefutás | 1a. A menteni kívánt kifejezés szintaktikailag helytelen (csak kifejezés mentése esetén)  2a. A megadott névvel már létezik fájl a mappában |

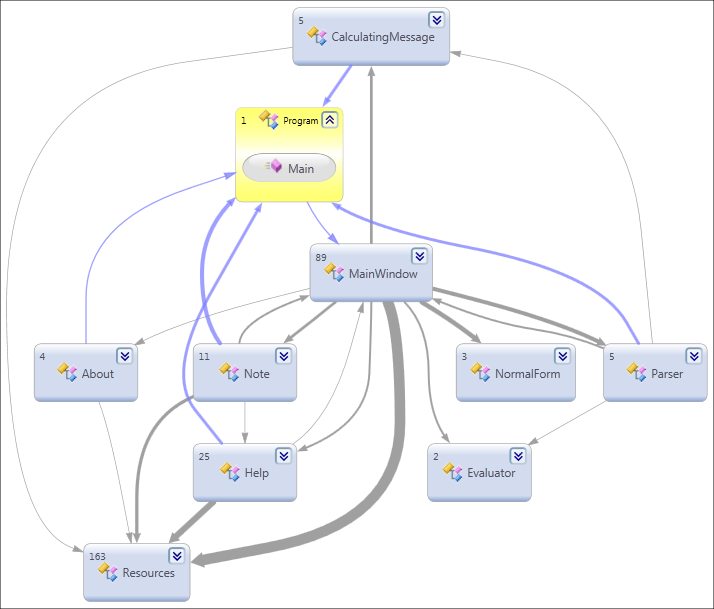
Adatok mentése: kifejezés, számítási eredmények vagy jegyzet fájlba mentése, a későbbi felhasználhatóság céljából. Ha az egész feladat mentését választjuk, akkor az összes eredmény mentésre kerül, saját fájltípusának megfelelően.

|  |  |
| --- | --- |
| **Használati eset** | **8. Adatok betöltése** |
| Leírás | Elmentett adatok megnyitása. |
| Érintett aktor | Függvény |
| Kiváltó esemény | Korábban mentett adatok betöltése újrafelhasználás céljából. |
| Előfeltételek | - |
| Sikeres lefutás | 1. A felhasználó rákattint a kívánt panelen a “Megnyitás” ikonra  2. A felhasználó kiválasztja a megnyitni kívánt fájt  3. A program beolvassa az adatokat a fájlból |
| Eredmény | A programban megjelennek a betöltött adatok. |
| Alternatív lefutás | 2a. A kiválasztott fájl nem érvényes |

Adatok betöltése: korábban fájlba mentett adatok újbóli megnyitása, mely lehet kifejezés (.xpr), igazságtábla (.xls) vagy jegyzet (.txt).

## 5.3. Az osztályokról

A program futása a „Program” osztály „Main”metódusának meghívásával kezdődik. Ebben a függvényben létrejön egy példány a „MainWindow”osztályból, amely a program érdemi működéséért felelős, a munka jelentős részét, valamint más osztályok helyes együttműködését is vezérli. Ezen példány létrejöttével jelenik meg a képernyőn a program főablaka. Ezt követően a felhasználói interakciótól függően további osztályok használatára is sor kerülhet. Az egyes osztályok használatát, más osztályokból való elérését a 9. ábra szemlélteti.



9. ábra. Osztályok és függőségeik

A következőkben röviden ismertetem minden osztály szerepét, illetve egy-egy jellemző esetet, melyben az osztály használatára sor kerül a program futása során.

„**Program**” osztály: a program indításakor van szerepe. Minden esetben, mikor elindítjuk a programot, ez az osztály létrehoz egy példányt a „MainWindow” osztályból.

„**MainWindow**” osztály: a program főablaka. A felhasználói felületet, a program kinézetét kezeli. A Venn-diagram, igazságtábla és egyéb panelek, grafikus komponensek megjelenítéséért és helyes adatokkal való feltöltéséért felelős. A program futása során végig használatban marad, feladata még a felhasználói események kezelése, valamint a megjelenés ezen információk alapján történő frissítése is.

„**Parser**” osztály: a logikai kifejezés karakterekre bontását és értelmezését végzi. Bármely kifejezés kiértékelése ezen osztály segítségével történik. Sikeres futása esetén (szintaktikailag helyes kifejezés) olyan listát ad tovább kiértékelésre az „Evaluator” osztálynak, amely a feldolgozott kifejezés karaktereit tartalmazza.

„**Evaluator**” osztály: a szoftver helyes működéséhez nélkülözhetetlen számításokat végző - legfontosabb - osztály. A „Parser” osztály által értelmezett kifejezés kiértékelését végzi, eldönti egy kifejezésről, hogy igaz vagy hamis. Minden kifejezés kiértékelése a „Parser” és „Evaluator” osztály egymás utáni alkalmazásával történik.

„**NormalForm**” osztály: a diszjunktív, valamint konjunktív normálformák számításáért felelős osztály. Egy kétdimenziós tömbből - mely a változók logikai értékeit és neveit, valamint a hozzájuk tartozó kiértékelt eredményeket tartalmazza - állítja elő a normálformákat.

„**CalculatingMessage**” osztály: feladata mindössze annyi, hogy a kifejezés kiértékeléséhez tartozó időigényes számítások elvégzése alatt blokkolja a felhasználói felületet, valamint tájékoztatja a felhasználót a háttérben zajló számításokról.

„**Note**” osztály: jegyzetek - kívánt méretben és pozícióban történő - megjelenítését, betöltését és mentését kezelő osztály.

„**Help**” osztály: a súgó funkciót és annak kezelését megvalósító osztály.

„**About**” osztály: a program névjegyét és az ahhoz tartozó információkat tartalmazza.

„**Resources**” osztály: automatikusan generált osztály, mely a szoftverben használt külső komponensek - melyek nem részei a .NET keretrendszernek -, például ábrák, ikonok, gombok alkalmazásához szükséges.

A fent ismertetett osztályok (az utolsó kivételével) az osztály nevével azonos nevű, „.cs” kiterjesztésű fájlokban vannak implementálva. Ezen fájlok megtalálhatók a mellékelt CD-n található „PROJECT\logika\_v1.0.0.zip” állomány „logika\_v1.0.0\logika” mappájában. A „Resources” osztály által felhasznált fájlok a kibontott állomány „logika\_v1.0.0\logika\Resources” mappájában találhatók.

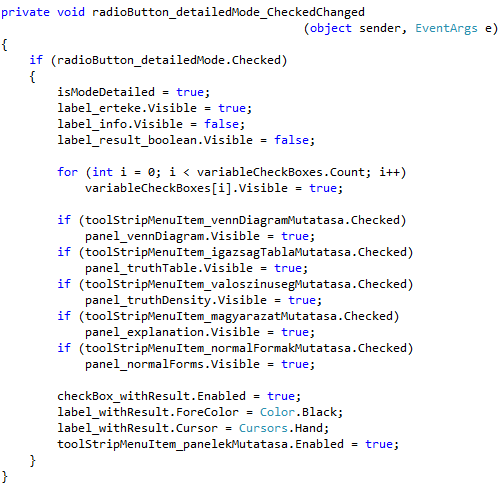
## 5.4. Eljárások implementálása, függvények

Ebben az alfejezetben a program néhány funkciójának implementációját ismertetem. A legfontosabb függvények egy része - összetettsége, valamint hosszúsága miatt - nem kerül itt bemutatásra, azonban a teljes forráskód - a fent említetthez hasonló módon - megtekinthető a mellékelt CD „PROJECT\logika\_v1.0.0.zip” állományának „logika\_v1.0.0\logika” mappájában, valamint megvalósításuk folyamatát a 4.2. alfejezetben ismertettem.

### 5.4.1. Kiértékelési mód váltás

Ha a felhasználó a program „Kiértékelési mód” panelén kiértékelési módot vált, az ezen eseményhez rendelt függvény hívódik meg, amely beállítja a program változóit, és módosítja a főablak elrendezését az aktuális beállítások, valamint a kiválasztott mód szerint. „Részletes” módot választva a 10. ábrán látható programrészlet hajtódik végre.

Megemlítendő az osztály *isModeDetailed* nevű logikai változója, amely „Részletes” mód esetén igaz, „Gyors” mód esetén pedig hamis értéket tárol. Ezt követően a változók logikai értékeit ábrázoló „dobozokat” tartalmazó *variableCheckBoxes* lista elemeit, majd a főablak paneleit jeleníti meg a program, a felhasználó beállításaitól függően. Ezek egyaránt az adott komponens *Visible* adattagjának igaz, illetve hamis értékekre állításával hajthatók végre.

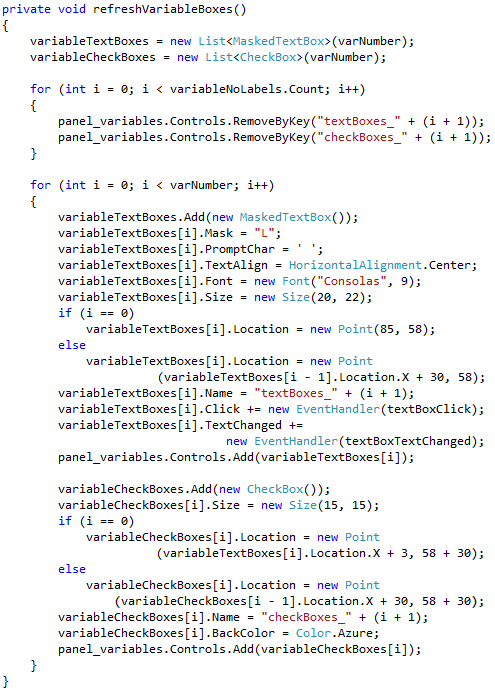


10. ábra. Kiértékelési mód váltáskor meghívódó függvény

### 5.4.2. Változók adatainak frissítése

Mivel a felhasználó futás közben tetszőlegesen változtathatja a változók számát, ezért szükség van egy eljárásra, ami ezen változtatásoknak megfelelően, futás közben frissíti a változókhoz tartozó komponenseket a képernyőn.

A 11. ábrán bemutatott függvény előbb eltávolítja, majd dinamikusan létrehozza a változók személyre szabásához szükséges *Control*-okat, a felhasználó által meghatározott számban.



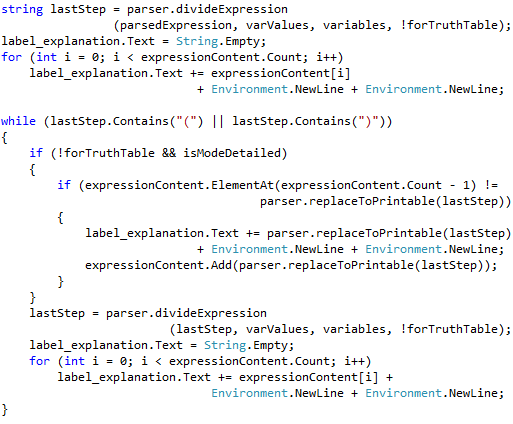
11. ábra. A „Változók” panel frissítéséért felelős metódus

Utasításaival a függvény beállítja többek közt a komponensek betűtípusát, méretét, pozícióját, valamint nevét is, a későbbi egyszerű eltávolítás érdekében. Mindezek mellett eseménykezelőket is rendel a kívánt komponensekhez, a komponensnév kívánt eseménye után feltüntetett „+= new EventHandler()” kulcsszavak megadásával.

### 5.4.3. Kifejezés kiértékelése

A megadott logikai kifejezés kiértékelése egy hosszú, sok függvényhívást tartalmazó folyamat. A „Parser” osztály „parseExpression” metódusa által sikeresen feldolgozott kifejezés zárójelezés alapján történő felbontását ugyanezen osztály „divideExpression” nevű eljárása végzi el.

A 12. ábrán látható kódrészletből látszik, hogy főosztály „evaluateExpression” függvénye addig hívja az imént említett „divideExpression” metódust, míg az tartalmaz zárójelet.



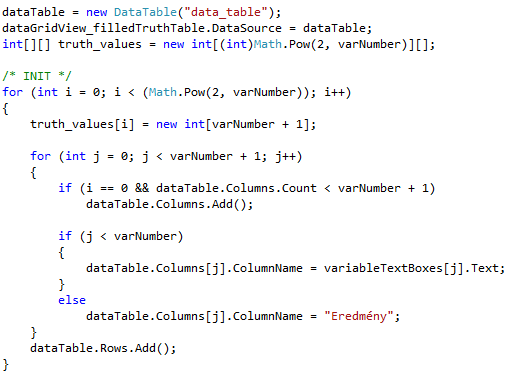
12. ábra. Az „evaluateExpression” függvény egy részlete

A futás során kapott részkifejezéseket az *expressionContent* változóban tárolja a program, a későbbi megjelenítés érdekében. A *forTruthTable* logikai változó azt mondja meg, hogy a függvény aktuális hívása az igazságtábla egy sorának kiszámolásához, vagy a „Változók” panelen meghatározott változóértékek alapján történő kiértékeléshez végez-e számításokat.

### 5.4.4. Igazságtábla felírása

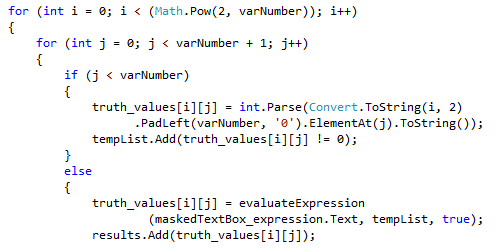
A kifejezéshez tartozó igazságtábla számolásának menetét már korábban ismertettem (4.2.2. pont). Most bemutatom a tábla létrehozásáért felelős „calculateTruthTable” nevű metódus lényeges lépéseit.

Miután a függvény meghívódik, elsőként maga az igazságtábla, valamint a hozzátartozó adattagok kerülnek beállításra. Az ezen műveletekért felelős kódrészletet a 13. ábra szemlélteti.



13. ábra. A „calculateTruthTable” metódus inicializáló kódja

A *dataTable* változó felel a tábla kezeléséért, és a kiszámolt adatok tárolásáért. A *dataGridView\_filledTruthTable* nevű *DataGridView* típusú komponens (5. ábra) teszi lehetővé az adatok megjelenítését. Ennek *DataSource* adattagjaként beállítjuk a fent említett *dataTable* táblát. Az igazságtábla tényleges értékeit a *truth\_values* nevű kétdimenziós tömbben fogjuk tárolni. Két ciklussal beállítjuk a tömb dimenzióit, illetve a *dataTable* komponens oszlopainak hozzáadása után hozzáadjuk a szükséges sorokat is. Ezt követi az adatok feltöltése (14. ábra).



14. ábra. A „calculateTruthTable” metódus adatfeltöltésért felelős részlete

Ha a *j* ciklusváltozó értéke kisebb, mint a változók számát tároló *varNumber* értéke, akkor bináris számmá alakítva *j*-t, majd számjegyeit a változók oszlopához rendelve megkapjuk a változók kombinációit az igazságtábla összes sorára (például az 5. ábra első 3 oszlopa esetén). Ha *j* értéke egyenlő *varNumber* értékével, meghívódik az „evaluateExpression” kiértékelő függvény, melynek paraméterei maga a kifejezés, az adott sorhoz tartozó változóértékek listája, valamint egy logikai igaz érték, amely azt jelzi, hogy a számolás az igazságtábla soraira vonatkozik. Értelemszerűen ennek a visszatérési értéke adja az adott sorhoz tartozó eredményt. Ha a ciklus végig ért az igazságtábla összes során, a kész tábla megjeleníthető a képernyőn.

# 6. Felhasználói ismertető

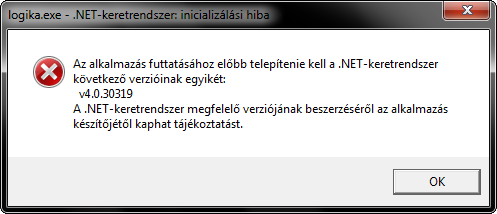
## 6.1. Előkészítés, környezet

A program fejlesztését és tesztelését Windows 7 operációs rendszer alatt, valamint .NET v4.0 környezetben végeztem. Az eredményes futtatáshoz ezért a .NET v4.0.30319 verziójára és az alábbi operációs rendszerek valamelyikére van szükség:

- Windows Vista

- Windows 7

Korábbi operációs rendszereken a program nem fog elindulni. A szükséges verziójú .NET része a Windows hivatalos frissítéseinek, így ha Windows Vista, vagy újabb operációs rendszert használunk, nagy valószínűséggel rendelkezünk a legfrissebb verzióval. Ha mégsem, a szükséges telepítő megtalálható a dolgozathoz mellékelt CD „PROGRAM\dotNET” nevű mappájában, így hiánya esetén nincs más dolgunk, mint feltelepíteni azt. Ha a keretrendszer v4.0.30319 verziójánál korábbi verzió van gépünkön, a program szintén nem fog elindulni, és az alábbi hibaüzenetet kapjuk:

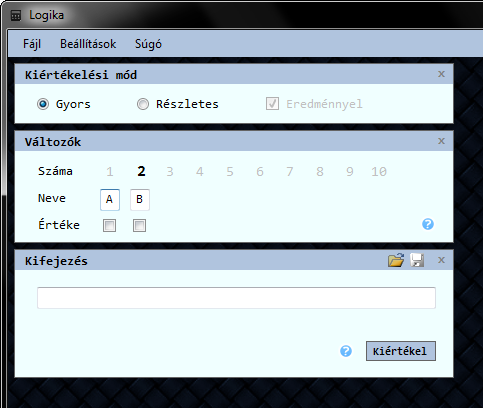


15. ábra. Hibaüzenet megfelelő verziójú .NET hiánya esetén

A szükséges verzió sikeres telepítését követően nincs több dolgunk, használatba vehetjük a szoftvert.

### 6.1.1. Futtatás

A program indításához nincs szükség telepítésre, mindössze a „logika.exe” nevű futtatható állományt kell elindítanunk, melyet a mellékelt CD „PROGRAM” nevű mappájában találunk. Miután elindult a program, a következő képernyő fogad minket:



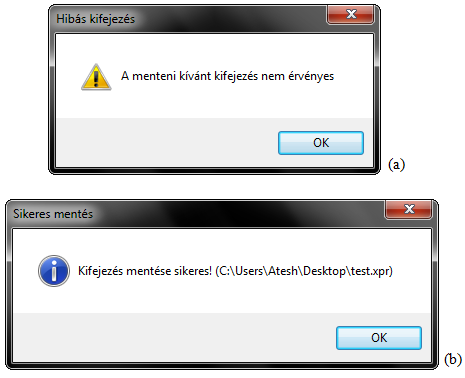
16. ábra. A program indítását követő képernyő panelei

## 6.2. A program használata

### 6.2.1. Indítás után

Az indítást követően 3 panel látható a képernyőn (16. ábra), melyek sorban a következők: kiértékelési mód, változók, kifejezés. Az első panelen lehetőségünk van kiértékelési módot váltani. A „Változók” panelen beállíthatjuk a kifejezésben használni kívánt változók számát, nevét, valamint logikai értékét. Egy változó értéke akkor igaz, ha az alatta található dobozt kipipáltuk. Ha nem tartalmaz pipát, a hozzá tartozó változó logikai értéke hamis. A „Kifejezés” panelen adhatjuk meg a kiértékelni kívánt kifejezést, melynek kiértékelését a „Kiértékel” gombra kattintva kezdeményezhetjük. Ha a kifejezés beírásával kapcsolatban segítségre van szükségünk, a „Kiértékel” gomb melletti kérdőjelre kattintva hasznos információkat kaphatunk (24. ábra). Fontos megemlíteni, hogy a kifejezés megadását követő kiértékelés után a program az egyes műveletek karaktereit minden esetben az adott művelet alapértelmezett - a 24. ábrán a műveletek oszlopainak legfelső sorában, vastag betűvel szereplő - karakterére cseréli (a kifejezés könnyebb átláthatósága érdekében), a szóközöket pedig elnyeli. Az implikáció (=>), valamint az ekvivalencia (<=>) operátorain kívül az összes többi művelet operátora egyetlen karakterből áll.

Amennyiben korábban elmentett kifejezésünket szeretnénk újra megnyitni, ezt a panel bezárása gomb mellett található mappa ikonra kattintva tehetjük meg. Mentéshez ugyanitt kattintsunk a lemez ikonra. A kifejezések elmentését a program saját fájlkiterjesztésbe végzi (.xpr). Mentés előtt a program ellenőrzi, hogy szintaktikailag helyes-e a menteni kívánt kifejezés, ha nem, figyelmeztet, és a mentés nem folytatódik (17/a ábra).



17. ábra. Üzenet érvénytelen (a) és érvényes (b) kifejezés mentése esetén

Ha a mentés sikeres, a kifejezést tartalmazó fájl létrejön a felhasználó által megadott mappában, melyről a program rövid üzenetben tájékoztat (17/b ábra). A mentett fájlt később a fent említett módon újra megnyithatjuk.

### 6.2.2. Gyors mód

„Gyors” módban történő kiértékelés esetén csupán a „Változók” panelen megadott logikai értékek alapján értékeli ki a program a kifejezést, majd megjeleníti a végeredményt, ami igaz vagy hamis lehet. Ha egy kifejezés gyors megoldására van szükségünk, és a végeredményen kívül semmi más információra nem vagyunk kíváncsiak, ez a mód tökéletesen megfelel igényeinknek.

### 6.2.3. Részletes mód

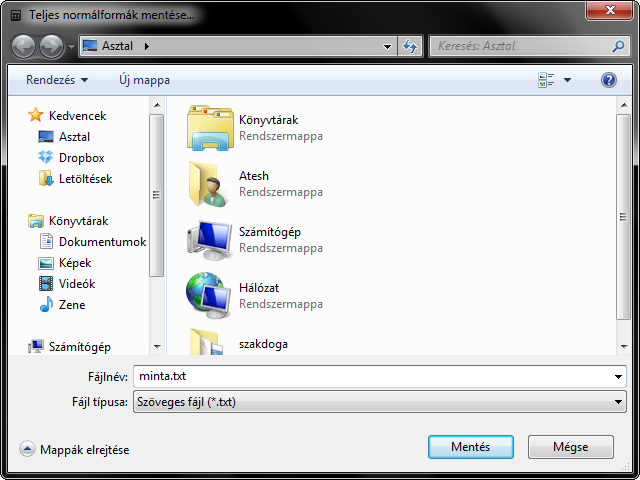
Ha a „Kiértékelési mód” panelen „Részletes” módra váltunk, az összes panel megjelenik. Ekkor az „Eredménnyel” mezővel kiválaszthatjuk, hogy a kifejezés igazságtábláját kitöltetjük-e a szoftverrel, vagy magunk szeretnénk kitölteni.



18. ábra. „Részletes” kiértékelési mód panelei

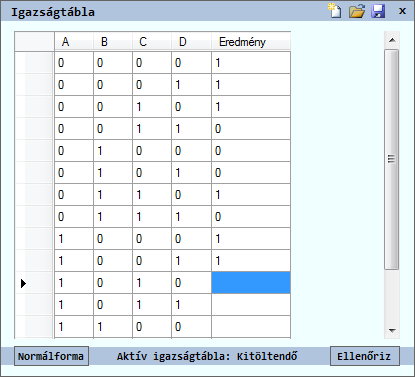
A „Teljes normálformák” panelen a kiértékelést követően megjelenik a kifejezéshez tartozó diszjunktív, illetve konjunktív normálforma (7 változóig). A panelen található lemez ikonra kattintva szintén lehetőség van mentésre. A normálformákat a program szövegfájlként (.txt) menti (19. ábra), így megnyithatók bármely egyszerű szövegszerkesztővel.

A „Magyarázat” panel a kiértékelés után a megoldás levezetésének főbb lépéseit tartalmazza, amely a mentés ikonra kattintva szintén sima szövegfájlként (.txt) elmenthető, a 19. ábrához hasonló módon.



19. ábra. Teljes normálformák mentése

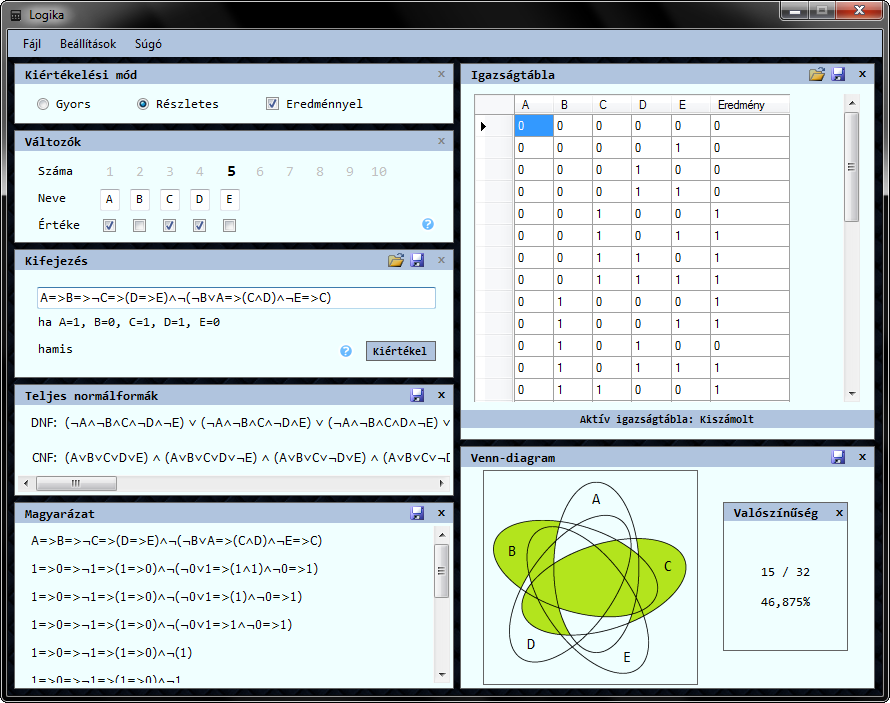
Az „Igazságtábla” panelen a kifejezéshez tartozó igazságtábla, vagy egy eredményeket nem tartalmazó, kitöltendő igazságtábla jelenik meg attól függően, hogy a felhasználó kiértékelés előtt a „Kiértékelési mód” panelen kipipálta-e az „Eredménnyel” felirat melletti dobozt. A kitöltött igazságtábla megjelenítése esetén a panel jobb felső sarkában található „Igazságtábla mentése” vagy „Igazságtábla megnyitása” gombokat használhatjuk az igazságtáblánk elmentésére, valamint megnyitására. Ehhez a program Excel munkafüzet fájlokat használ (.xls). Fontos megjegyezni, hogy sok változó esetén (7 vagy több) a mentés akár néhány másodpercet is igénybe vehet. Ugyancsak ebben az esetben, ha a változók oszlopa előtti oszlopra kattintva a tábla egyes sorait kijelöljük (a Ctrl gomb lenyomva tartásával akár egymástól távol eső sorokat is), a Venn-diagramon a kijelölt sorokhoz tartozó halmazterületek villogni kezdenek, ezzel is szemléltetve a két ábrázolásmód közti összefüggéseket. Ha a kitöltésre váró igazságtáblát választottuk ki, további vezérlők jelennek meg a tábla alatt (20. ábra), melyek a „Normálforma” és az „Ellenőriz” gombok. Előbbire kattintva az általunk tetszőlegesen kitöltött táblához készíttethetünk normálformákat a programmal, melyek a „Teljes normálformák” panelen jelennek meg, akárcsak a kifejezés „Részletes” módban történő kiértékelése esetén. Az „Ellenőriz” gombra kattintva ellenőriztethetjük az általunk - egy megadott kifejezés alapján - kitöltött igazságtáblát a program által készítettel, ám ebben az esetben értelemszerűen szükség van egy kiértékelt kifejezésre. Ha ez nem áll rendelkezésre, a program ezt hibaüzenettel jelzi, a 17/a ábrához hasonló módon.



20. ábra. Kitöltendő igazságtábla

A panel jobb felső sarkában található „Új igazságtábla” ikonra kattintva újabb kitöltendő igazságtáblát hozhatunk létre, illetve az alul található gombok közötti szövegre kattintva váltogathatunk a kitöltött és a kitöltendő táblák között.

A „Venn-diagram” panel a kiértékelt kifejezés megoldását tartalmazza Venn-diagramon ábrázolva (5 változóig), valamint egy belső panelt, amelyben a feladatot eseményalgebraként értelmezve az esemény bekövetkezési valószínűségét találjuk, a 3.3.3. pontban leírtak alapján. A diagram elmentésére a „Venn-diagram mentése” ikonra kattintva van lehetőség. Ezt követően a program elmenti a Venn-diagramot a kiválasztott mappába, PNG fájlformátumban.

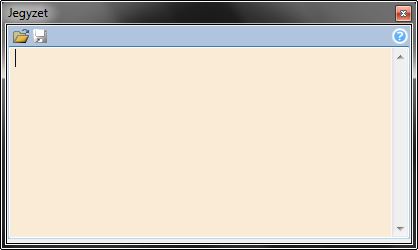


21. ábra. Részletes kiértékelést követő képernyő

„Részletes” módban lehetőség van a panelek elrejtésére, melyet az adott panel jobb felső sarkában található bezárás gombbal, vagy a „Beállítások” menü „Panelek mutatása” menüpontjában található lehetőségek valamelyikével hajthatunk végre. Ez a menüpont tartalmaz egy „Mindet mutat”, valamint egy „Mindet elrejt” menüpontot is, az egyszerűbb kezelés érdekében.

### 6.2.4. Jegyzetek

A szoftver használata során lehetőség van feljegyzések készítésére is. Ilyen jegyzetet a „Fájl” menü „Új jegyzet” parancsával hozhatunk létre. Ha egy korábbi feljegyzést szeretnénk megnyitni a programban, ezt a „Fájl” menü „Jegyzet megnyitása” menüpontjával tehetjük. Ezen lehetőségeket a 23. ábra szemlélteti.

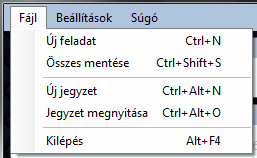


22. ábra. Üres jegyzet

A „Jegyzet” ablak bal felső sarkában található megnyitás és mentés ikonokkal nyithatunk meg korábban elmentett jegyzetet, valamint készíthetünk új mentést (22. ábra). A program a jegyzeteket hagyományos szövegfájlként (.txt) kezeli, így azok bármely egyszerű szövegszerkesztőben megnyithatók. Lehetőség van továbbá az ablak bezárás gombja alatt található kérdőjelre kattintva a program súgójában a jegyzetekre vonatkozó rész megnyitására. A program futása során tetszőleges számú jegyzet hozható létre, nyitható meg és lehet használatban, akár egyidőben is.

### 6.2.5. Menük

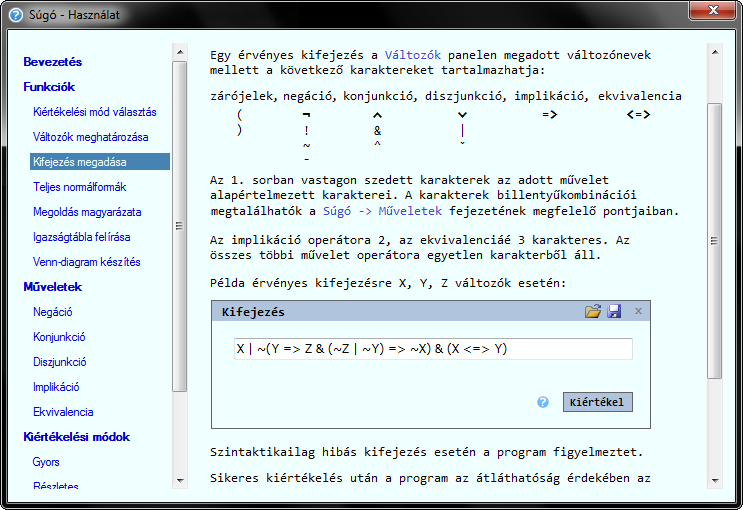
A program rendelkezik egy menüvel, ahol a legfontosabb műveletek érhetők el. A „Fájl” menüpont alatt új feladat kezdésére, feladat mentésére, jegyzet készítésére és megnyitására, valamint kilépésre van lehetőség, ezenkívül minden opció mellett megtalálható a hozzá tartozó billentyűkombináció is (23. ábra). Az „Új feladat” parancs hatására a program felajánlja az aktuális feladat mentését, ha van használható eredmény, ezt követően a korábbi eredmények és a panelek tartalma törlésre kerül, a program visszaáll alaphelyzetbe, lehetőséget biztosítva új feladat megoldására. Az „Összes mentése” menüpont kizárólag „Részletes” módban elérhető, illetve szükséges, hogy az összes panel tartalmazzon eredményt. Ebben az esetben ezt a menüpontot használva, valamint a feladat nevét és a mentés helyét megadva a program 5 különböző fájlba menti a különböző panelek eredményeit, a 6.2.3. pontban leírtak szerint, a mentéseket sorban végrehajtva. Az „Új jegyzet” menüpontot kiválasztva jegyzetet hozhatunk létre, vagy ha egy korábban mentett feljegyzést szeretnénk megnyitni, azt a „Jegyzet megnyitása” paranccsal érhetjük el. A „Kilépés” menüponttal a program - ha van menthető eredmény - a feladat mentésének felajánlása után bezárul.



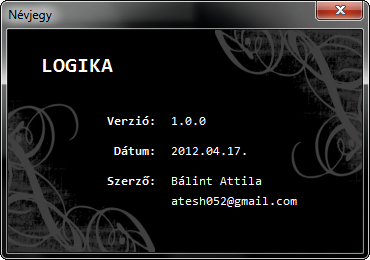
23. ábra. A „Fájl” menü

A „Beállítások” menüben a változók automatikus kitöltését, valamint a panelek láthatóságát állíthatjuk be. Ha az automatikus kitöltést engedélyezzük, a változók számának módosítása esetén a program automatikusan feltölti a változók neveit ábécé sorrendben, „A”-val kezdődően. Ha nem engedélyezzük, minden egyes változó nevét nekünk kell megadni.

A „Súgó” menüpont alatt a szoftver használatához találhatunk segítséget kategóriák szerint rendezve rövid, de hasznos információkkal és képernyőképekkel illusztrálva (24. ábra), valamint megtekinthetjük a program névjegyét is (25. ábra).



24. ábra. A program súgójának „Kifejezés megadása” című oldala.



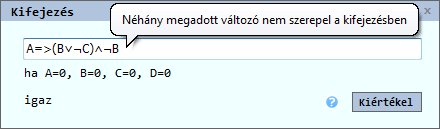
25. ábra. A program névjegye

# 7. Futtatási tapasztalatok

A következőkben röviden ismertetek néhány, a program segítségével kiértékelt kifejezést, valamint a számítások lefutásához szükséges időket. A futtatás során törekedtem arra, hogy a teszteket lehetőség szerint változatos adatokkal végezzem, valamint az eredmények összevethetőségét szem előtt tartva a programot végig „Részletes” módban használtam. Az összes kifejezés, melyet a tesztek során kiértékeltem, valamint a hozzájuk tartozó eredmények megtalálhatók a dolgozathoz mellékelt CD „TESTS” mappájában. Fontos megjegyezni, hogy az elvégzett tesztek mindegyikét Windows 7 operációs rendszer alatt, tesztenként 10 alkalommal futtatva végeztem, a .NET keretrendszer jelenleg elérhető legfrissebb stabil, 2010 áprilisában kiadott (v4.0.30319) verziójával.

## 7.1. Elvégzett tesztek

A számítási igény korábban említett (3.4.2. pont) exponenciális növekedése miatt a változók számának növelése döntő hatással van a program futási idejére, amely így *O*(2*n*) ≈ 2*n*. Mivel a változók számának és egyéb jellemzőinek meghatározását teljes mértékben a felhasználóra bíztam, nem tartottam szükségesnek, hogy a program kiszűrje a definiált, de fel nem használt (fiktív) változókat. Mindemellett fontosnak tartottam azt az esetet is számításba venni, mikor figyelmetlenség, vagy egyéb hasonló okból adódóan marad el egyes beállított változók felhasználása, ezért a program egy felugró üzenetben figyelmeztet erről (26. ábra). Így a felhasználó pontosan az általa megadott paraméterek szerinti eredményt kapja, azonban mégis sikerül kiszűrni a figyelmetlenségből adódó hiányosságokat.



26. ábra. Figyelmeztetés fel nem használt változók esetén

### 7.1.1. Teszt 1

Elsőként egy egyszerűbb kifejezés, a 3.4. alfejezetben vizsgált

( *A  B  C* )  ¬*B*  ¬*C*

kiértékelését végeztem el 3 változó esetén. Ebben az esetben a kiértékelés átlagosan 0.31 másodpercet vett igénybe.

Ezután a változók számát 5-re állítottam, a szükséges változókat kicseréltem, a kifejezés műveleteit azonban nem változtattam. Így az

( *A  B  C* )  ¬*D*  ¬*E*

kifejezés megoldásához a program több mint kétszer annyi időt használt fel. A kiértékelés elindítása és az eredmény megjelenítése között eltelt idő átlagosan 0.79 mp volt.

### 7.1.2. Teszt 2

A folytatásban egy összetettebb kifejezés kiértékelését vizsgáltam. Első alkalommal 3 változóval a

( *C*  ¬*B  A =*>( ¬*C*   *B* ) ) <=> ( ( *A* <=> ¬*B  C* ) =>

( *A  C* ) ** ( *B*  ¬*C* ) )  ¬*A*

kifejezést értékeltem ki, ebben az esetben a futási idők átlaga 0.32 másodperc volt.

Ezt követően 5 változósra bővítve a

( *C*  ¬*B  A =*>( ¬*E  D* ) ) <=> ( ( *C* <=> ¬*B  D* ) =>

( *E  A* ) ** ( *E*  ¬*B* ) )  ¬*C*

kifejezést, a kiértékelés átlagosan 1.41 másodpercbe telt, ami több mint négyszeres lassulást jelent.

Következő lépésként a kifejezést 7 változósra bővítettem, így a

( *C*  ¬*B  A =*>( ¬*E*   *D* ) ) <=> ( ( *F* <=> ¬*B  G* ) =>

( *A  D* ) ** ( *E*  ¬*F* ) )  ¬*C*

kifejezés kiértékelése 0.88 másodpercet vett igénybe.

Végül ugyanezt a kifejezést vizsgáltam 10 változóra, ez esetben a

( *C*  ¬*B  A =*>( ¬*E*   *D* ) ) <=> ( ( *F* <=> ¬*H  G* ) =>

( *C  B* ) ** ( *A*  ¬*J* ) )  ¬*I*

megoldásához szükséges átlagos idő 2.29 másodperc volt.

### 7.1.3. Teszt 3

Utolsóként egy összetett kifejezés kiértékelését végeztem el, ezúttal még részletesebben, 3, 5, 7 és 10 változó esetén. Első esetben a

( ¬*A*  ¬*B* => *C* )  ( *C* <=> ( *B* ¬*A* => ¬*C* ( ( ( *B*  *A* ) => ¬*B* )=>

( ¬*C  A* ) ) )  ¬*B )  A* => *C*  ( ¬*B * ( *A* => *C*  *B* )) <=> ¬*C*

kiértékelését a program átlagosan 0.34 másodperc alatt végezte el.

A fenti kifejezést 5 változóra bővítve a

( ¬*A*  ¬*B* => *D* )  ( *C* <=> ( *A* ¬*E* => ¬*C* ( ( ( *D*  *A* ) => ¬*B* )=>

( ¬*C  A* ) ) )  ¬*E )  C* => *D *  ( ¬*B * ( *A* => *C*  *E* )) <=> ¬*D*

kiértékelésével a program 1.08 másodperc alatt végzett átlagosan.

7 változóval a

( ¬*A*  ¬*B* => *D* )  ( *C* <=> ( *G* ¬*E* => ¬*F* ( ( ( *D*  *A* ) => ¬*B* )=>

( ¬*C  F* ) ) )  ¬*E )  A* => *D*  ( ¬*B * ( *F* => *C*  *E* )) <=> ¬*G*

kifejezés kiértékeléséhez szükséges átlag idő 0.99 másodperc volt.

Végül 10 változós esetben a

( ¬*A*  ¬*B* => *D* )  ( *C* <=> ( *J* ¬*E* => ¬*F* ( ( ( *D*  *H* ) => ¬*G* )=>

( ¬*I  E* ) ) )  ¬*D )  G* => *I*  ( ¬*B * ( *F* => *C*  *H* )) <=> ¬*J*

kifejezés megoldását a program 3.21 másodperc alatt állította elő.

## 7.2. Levont következtetések

A fenti tesztek legfontosabb információit táblázatokba foglaltam, melyek lejjebb találhatók. Ezek alapján egyértelműen megállapítható, hogy a feladat megoldásához szükséges idő döntő mértékben a változók számától függ, hiszen *n* változó esetén a program 2*n* alkalommal hívja meg a kifejezést kiértékelő függvény-sorozatot. Mivel egy összetett kifejezés egyszeri kiértékelése (egy sor az igazságtáblában) nem tart nagyságrendekkel hosszabb ideig egy egyszerű kifejezés egyszeri kiértékelésénél, így világossá válik a különbség. Ebből következően egy jelentősen összetett, azonban kevesebb változókból álló kifejezés esetében a megoldás folyamata jóval kevesebb ideig tart, hiszen hiába a hosszabb kifejezés, a kiértékelés kevesebb alkalommal ismétlődik. Elmondhatjuk tehát, hogy egy kiértékelés megkezdése előtt célszerű pontosan felmérni, hány változóra is lesz majd szükségünk, és ezek alapján meghatározni a változók számát a programban. Még ha nem is használjuk fel kifejezésünkben az összes megadott változót, a futási időt ez nem rövidíti, azaz gyakorlatilag megegyezik azzal az esettel, mikor minden változót felhasználunk, hiszen a függvényhívások ugyanúgy végrehajtódnak.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Teszt 1** | | **Teszt 2** | | | |
| **Kiértékelési mód** | Részletes | | Részletes | | | |
| **Kifejezés** | Egyszerű | | Közepes bonyolultságú | | | |
| **Változók száma** | 3 | 5 | 3 | 5 | 7 | 10 |
| **Futási idő (mp)** | 0.31 | 0.79 | 0.32 | 1.41 | 0.88 | 2.29 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Teszt 3** | | | | |
| **Kiértékelési mód** | Részletes | | | | |
| **Kifejezés** | Összetett | | | | |
| **Változók száma** | 3 | 5 | 7 | 10 | 10 (3) |
| **Futási idő (mp)** | 0.34 | 1.08 | 0.99 | 3.21 | 3.03 |

A 2. és 3. teszt során végzett mérések közül szembetűnő, hogy a kifejezés 5 változós eseteinek futási ideje hosszabb, mint 7 változó esetén. Ennek magyarázata, hogy a program 5 vagy kevesebb változó használata esetén Venn-diagramot is készít, így ezen esetek futási idejét a diagram készítésének folyamata is lassítja. A 3. teszt utolsó méréseként elvégeztem a kifejezés kiértékelését 10 megadott változóra úgy, hogy ebből csak 3 változót használtam a kifejezésben. A futási idő jól láthatóan közel azonos a 10 felhasznált változós esethez, a fent említettek okok miatt.

A feladatok mentésére felhasznált időből pontos méréseket nem végeztem, de megállapítható, hogy a feladat összes komponensének mentése közül az igazságtábla mentése történik a leglassabban. Itt az Excel munkalap létrehozását követően az igazságtábla fájlba exportálása veszi igénybe a legtöbb időt. Értelemszerűen csak nagy változószám esetén lesz a folyamat másodperces nagyságrendű. Méréseim során 10 változó esetén a mentési idő többször meghaladta a 8 másodpercet is.

## 7.3. Kiegészítés

A tesztekből világosan látszik tehát, hogy a program által a kifejezés kiértékelésére fordított idő gyakorlatilag teljes mértékben a meghatározott változók számától függ. Egy további tesztet elvégezve azonban megállapítottam, hogy jelentősen összetettebb 10 változós kifejezés esetén (melyet a mellékelt CD „TESTS” mappájában található „test4\_extra.xpr” nevű fájl tartalmaz) a futási idő lényegesen lassabb, akár perces nagyságrendű is lehet. Ez azt jelenti tehát, hogy egy átlagos, bonyolultnak is nevezhető (például Teszt 3) kifejezés szintjén nem jelentkezik lényeges lassulás a kifejezés összetettségéből adódóan, ennél jóval nagyobb összetettségű kifejezés esetén azonban már igen. A program használata során azonban szinte bizonyos, hogy ilyen bonyolultságú feladatokkal nem találkozunk. Ezek alapján az esetek döntő többségéről elmondható, hogy a kifejezés kiértékelése akár 10 változó esetén is megközelítőleg maximálisan 5-7 másodpercet vesz igénybe.

Jelentősebb futási időbeli különbségek jelentkezhetnek azonban az egyes számítógép konfigurációk teljesítményéből adódóan. A tesztekhez használt konfiguráció főbb jellemzői: Core 2 Duo T7300 CPU (2 mag, 2 GHz órajel), 1 GB RAM (DDR2). Ennél nagyobb teljesítményű gépen a futási idők értelemszerűen csökkenhetnek, míg egy gyengébb teljesítményű gép esetén akár jelentős lassulás is jelentkezhet.

# *Irodalomjegyzék*

[1] Birkhoff, G. (1940). Lattice theory. *American Mathematical Society Colloquium Publications*, 25(3).

[2] Boole, G. (1854). An investigation of the laws of thought. *Walton & Maberly*.

[3] Cohen, P. J. (1966). Set Theory and the Continuum Hypothesis. *Addison-Wesley*.

[4] De Morgan, A. (1847). Formal Logic, or The Calculus of Inference, necessary and probable. *Taylor & Walton*.

[5] Frege, G. (1879). Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.

[6] Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*,38: 173-198.

[7] Horváth, M., Joó, I. és Szalkai, I. (1991). A „Banach-elv”-ről. *Matematikai Lapok*,34(4): 253-300.

[8] http://hu.wikipedia.org/wiki/Boole-algebra. Wikipédia Boole-algebra (le-töltés dátuma 2012. április 12.). *Wikipédia információ*.

[9] http://hu.wikipedia.org/wiki/Matematikai\_logika\_t%C3%B6rt%C3%A9nete. Wikipédia Matematikai logika története (letöltés dátuma 2012. április 12.). *Wikipédia információ*.

[10] http://hu.wikipedia.org/wiki/Prolog. Wikipédia Prolog (letöltés dátuma 2012. április 12.). *Wikipédia információ*.

[11] Márkus, G., Márkusz, Zs. (1988). Logic puzzles and logic programming. *MTA SZTAKI*, *Tanulmányok*,207.

[12] Peirce, C. (1884-85). On the Algebra of Logic: A Contribution to the Philosophy of Notation. *American Journal of Mathematics*,7: 180-202.

[13] Russel, B. (1897). An Essay on the Foundations of Geometry. *Cambridge University Press*.

[14] Schröder, E. (1890-1905). Vorlesungen über die Algebra der Logik.

[15] Skolem, T. (1922). Axiomatized set theory.

[16] Stone, M. H. (1936). The Theory of Representations for Boolean Algebras. *Transactions of the American Mathematical Society*, 40: 37-111.

[17] Szalkai, I. (2001). Diszkrét matematika és algoritmuselmélet alapjai. *Veszprémi Egyetemi Kiadó*, 3-19.

# *Mellékletek*

A dolgozathoz mellékelt CD a következő mappákat tartalmazza:

- **FIGURES** mappa: a dolgozatban található ábrák gyűjteménye.

- **PROGRAM** mappa: a futtatható program és a szükséges .NET telepítője.

- **PROJECT** mappa: a program teljes forráskódja.

- **TESTS** mappa: a 7. fejezet tesztjeihez használt kifejezések és eredményeik.

- **THESIS** mappa: a dolgozat digitális formában.

- **URLS** mappa: az irodalmi rész internetes hivatkozásainak letöltött anyagai.

A mappák tartalma:

- **FIGURES** mappa tartalma:

- 01\_table.png

- 02\_diagram.png

- 03\_evaluation1.png

- 04\_evaluation2.png

- 05\_datagridview.png

- 06\_empty\_sets.png

- 07\_sets.png

- 08\_sets\_result.png

- 09\_class\_dependencies.png

- 10\_detailed\_mode\_checked\_changed.png

- 11\_refresh\_variables.png

- 12\_evaluate\_expression.png

- 13\_init\_truthtable.png

- 14\_fill\_truthtable.png

- 15\_required\_dotnet.png

- 16\_starting\_screen.png

- 17\_save\_expression.png

- 18\_detailed\_panels.png

- 19\_save\_dialog.png

- 20\_truth\_table.png

- 21\_detailed\_mode\_evaluation.png

- 22\_note.png

- 23\_file\_menu.png

- 24\_help.png

- 25\_about.png

- 26\_unused\_variables.png

- 27\_folder\_tree.png

- **PROGRAM** mappa tartalma:

- dotNET mappa (a .NET keretrendszer telepítője)

- logika.exe

- **PROJECT** mappa tartalma:

- logika\_v1.0.0.zip (Visual Studio 2010 project becsomagolva)

- **TESTS** mappa tartalma:

- Test1\_results mappa (7.1.1. alfejezet teszteredményei)

- Test2\_results mappa (7.1.2. alfejezet teszteredményei)

- Test3\_results mappa (7.1.3. alfejezet teszteredményei)

- test1\_vars3.xpr

- test1\_vars5.xpr

- test2\_vars3.xpr

- test2\_vars5.xpr

- test2\_vars7.xpr

- test2\_vars10.xpr

- test3\_vars3.xpr

- test3\_vars5.xpr

- test3\_vars7.xpr

- test3\_vars10.xpr

- test3\_vars10\_used3.xpr

- test4\_extra.xpr

- **THESIS** mappa tartalma:

- xdsi73.docx

- xdsi73.pdf

- **URLS** mappa tartalma:

- Boole-algebra\_files mappa

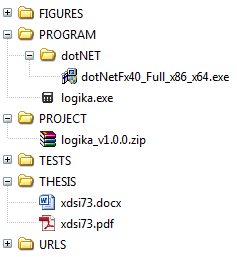
- Matematikai logika története\_files mappa

- Prolog\_files mappa

- Boole-algebra.html

- Matematikai logika története.html

- Prolog.html



27. ábra. A mellékelt CD mappaszerkezete