



MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Matematikai formulák leírásának egyszerűsítése DocBook környezetben

Török András

III. éves gazdaságinformatikus hallgató

Pribula Péter Balázs

III. éves programtervező informatikus hallgató

Konzulens:

Dr. habil Kovács László

egyetemi docens

Általános Informatikai Tanszék

Miskolc, 2011

Matematikai formulák leírásának egyszerűsítése DocBook környezetben
Simple mathematical formula language in DocBook environment

Kézirat lezárása:

Török András
Pribula Péter Balázs

Matematikai formulák leírásának egyszerűsítése DocBook környezetben

A dolgozat témaköre az informatika területén belül egy matematikai formulákat leíró nyelv létrehozatala az XML alapú DocBook XML nyelvhez.

A *DocBook* egy szemantikai jelölőnyelv, mely lehetővé teszi, hogy a dokumentum tartalmát formátum semleges formában tárolhassuk, amely kijelöli a logikai szerkezetet és a tartalmat. A DocBook szabvány lehetővé teszi a matematikai formulák beágyazását is a MathML szabvány alkalmazásával.

A DocBook alternatívái között szerepel a *LaTeX*, amely egy TeX-en alapuló szövegformázó rendszer. Nagy előnye, hogy a LaTeX-ben szereplő függvény könyvtárak, csomagok funkciók széles skáláját biztosítják. Amennyiben HTML formátumra konvertáljuk a dokumentumunkat, a képletek képekké konvertálódnak, ezzel romlik a minőség, nehézkes a frissítés, módosítás.

Az új *Microsoft Word* szerkesztők már matematikai képleteket meglepően jól tudnak kezelni, de számos hiányossággal is küzdenek, minthogy a képletek megjelenítésébe nem szólhatunk bele, kötött a szerkesztő, a konvertálás során a képletek minősége romlik és az interneten nem terjeszthető jól.

A *Mathematical Markup Language* (MathML, matematikai leíró nyelv) egy XML alapú, matematikai képletet leíró nyelv. Célja a matematikai formalizmust szabványos módon beágyazni a világhálón található dokumentumokba. A MathML megjelenítése már a böngészők által is elérhetővé váltak (pl.: Firefox 7). A MathML hiányossága, hogy bár logikusan építi fel a nyelvezetét, mégis a kódolása nehézkes, bonyolult az átláthatósága, hosszú kód keletkezik, ezért egy MathML kód elkészítése viszonylag időigényes.

Célunk az, hogy egy, a MathML bonyolultsági korlátait, megszüntető alternatív megoldást hozzunk létre a matematikai formulák kezelésére a DocBook nyelvhez. A javasolt megoldás a hosszú, s nehezen átlátható formalizmust egyszerűsíti, a kódolás idejét lerövidíti, szem előtt tartva a személyre szabhatóságot és kényelmi funkciókat.

Munkánk során létrehoztunk egy leíró nyelvet (prefix formula) a matematikai formulák leírásához és megalkottuk az ehhez szükséges szabályrendszert. A nyelv jelölésrendszere függvény-szerű, beszélő neveket használ, melyek tetszőlegesen testre szabhatóak. Egy *helyesen formált formula* jól mutatja a precedencia szinteket, és könnyen konvertálható egy fa szerkezetté, mely alapján a fordító egyszerűen elő tudja állítani a megfelelő MathML kódot. A fordító szótára tetszőlegesen szerkeszthető, bővíthető a felhasználó által a megfelelő szabályok betartásával.

Török András
Pribula Péter Balázs

Simple mathematical formula language in DocBook environment

The topic of this thesis is to create a simple descriptive language to describe mathematical formulas in DocBook XML environment.

The *DocBook* is a semantic markup language which allows users to represent the content of a document in a natural format, describing the logical structure as well as the content. *MathML* formulas can be embedded in the new versions of DocBook.

LaTeX is a typesetting system based on TeX. LaTeX has a great advantage by the wide range of packages and format files integrated in the system providing variety of features. By converting the document to HTML, the mathematical formulas are converted to images, which mean decreased quality of these formulas and to update of the document more difficult.

The new versions of *Microsoft Word* can handle mathematical formulas well, but there have been many difficulties, such as users cannot change the display style of the formula, or the qualities of the formulas deteriorate by converting the document.

The *Mathematical Markup Language* (MathML in short) is an XML based language for describing mathematical formulas. The goal of this language is to embed mathematical formulas into documents on the World Wide Web. The rendering of the MathML code is available in web browsers too (e.g.: Firefox 7). Though, MathML code is constructed in a very logical way, but it is difficult to write code, and the produced code is long and complicated, which makes the writing time-consuming.

Our aim is to eliminate the deficiency of MathML, by partly replacing, simplifying the difficult formalism, shorten the time of coding, bearing in mind the personalization and convenience features.

In our research we created a descriptive language (*prefix formula*) to describe the mathematical formulas, as well as the set of rules of this formal language. The notation of the language is function-like, the function names can be personalized at will. A well-formed prefix formula shows the levels of precedence, and can be converted into a tree easily. The conversion of a tree to MathML is simple. The compiler's dictionary can be extended easily.

TARTALOMJEGYZÉK

1.	Bevezetés	1
2.	Elektronikus dokumentumszerkesztési formátumok és módszerek áttekintése	2
2.1.	LaTeX	2
2.2.	WYSIWYG szerkesztők	3
2.3.	DocBook	4
2.4.	Összefoglalás	5
3.	Elektronikus dokumentumszerkesztők áttekintése	7
3.1.	Microsoft Word	7
3.2.	Oxygen	8
3.3.	XMLSpy	9
3.4.	Integre MathML Equotation Editor	9
4.	Áttekintés a reguláris nyelvekről	11
4.1.	Formális nyelvek alapjai	11
5.	Matematikai képletszerkesztés eszközeinek áttekintése	14
5.1.	A LaTeX formalizmusa	14
5.2.	MathML nyelv	14
6.	Saját formula a jelölő nyelv kidolgozására	15
6.1.	A formula leíró prefix nyelv bemutatása	15
6.2.	A prefix nyelv szabályai	17
7.	Átalakítási elvek	19
7.1.	Prefix fa	19
7.2.	Átalakítási alapelvek	19
7.2.1.	Matematikai képlet átalakítása prefix formulává	19
7.2.2.	Prefix formula átalakítása prefix fává	20
7.2.3.	Prefix fa átalakítása MathML	21
8.	Mintarendszer implementációja	22
8.1.	Fejlesztői környezet, implementáció nyelve	22
8.2.	A szótár szerkezete	23
8.3.	Az alkalmazás implementációs szerkezete	25
8.4.	prefix.tree	26
8.5.	prefix.dictionry	30
8.6.	prefix.parser	36
8.7.	Használati útmutató	37
9.	Tesztelés	40
10.	Összegzés és kittekintés	44
10.1.	Lehetőségek	45
10.2.	Oxygen plugin	45
11.	Irodalomjegyzék	48
	Mellékletek	1

TÁBLAJEGYZÉK

Összehasonlító táblázat	5
A <i>Node</i> osztály attribútumai	27
A <i>Node</i> osztály metódusai	27
A <i>PrefixTree</i> attribútumai.....	28
A <i>PrefixTree</i> metódusai.....	28
A <i>PrefixString</i> osztály attribútumai	29
A <i>PrefixString</i> osztály metódusai	29
A <i>TreeBuilder</i> attribútumai	29
A <i>TreeBuilder</i> metódusai.....	30
A <i>PrefixDictionaryRecord</i> osztály attribútumai	31
A <i>PrefixDictionaryRecord</i> osztály metódusai	32
A <i>PrefixDictionaryParser</i> osztály attribútumai.....	34
A <i>PrefixDictionaryParser</i> osztály attribútumai.....	35

ÁBRAJEGYZÉK

1. ábra A Microsoft Word 2010 felhasználói felülete.....	8
2. ábra Az Integre grafikus felhasználói felülete	10
3. ábra Példa a Presentation és a Content Markup MathML-ekre	15
4. ábra Prefix formula viszonya a beszélt nyelvhez és a LaTeX-hez.....	16
5. ábra A prefix nyelvet generáló nyelvtan produkciós szabályai	17
6. ábra Egy egyszerű példa, levezetés	18
7. ábra A példa levezetési fája	18
8. ábra A Prefix fa felépítése	19
9. ábra Példa a prefix formula prefix fává alakítására	21
10. ábra Az alkalmazás környezete	23
11. ábra Szerkezeti felépítés	26
12. ábra prefix.tree csomag osztálydiagramja	26
13. ábra prefix.dictionary csomag osztálydiagramja	31
14. ábra prefix.parser csomag osztálydiagramja.....	36
15. ábra Egy szótár bejegyzés mintája.....	37
16. ábra Sikeres futtatás 1.....	38
17. ábra A mellékelt fájlok elhelyezkedése	39
18. ábra Futtatás parancssorból	39
19. ábra Sikeres futtatás 2.....	40
20. ábra Hibaüzenet	40
21. ábra A teszt futási eredménye.....	41
22. ábra A bemeneti DocBook dokumentum képe	41
23. ábra A bemeneti DocBook dokumentum forrása.....	42
24. ábra Az elkészített dokumentum képe	42
25. ábra A kimeneti dokumentum forrása (részlet)	43
26. ábra A <i>test.outputxml.xml</i> PDF-re konvertált alakja.....	43
27. ábra A <i>test.outputxml.xml</i> XHTML-re konvertált alakja (Mozilla Firefox 7 böngészőben).....	44

1. BEVEZETÉS

Az informatikában a dokumentumkezelés fontos jelentőséggel bír. Napjainkban az XML alapú dokumentumábrázolás elterjedté vált a könnyű megérthetőségének, az egyszerűségének és az átfogó, minden sémára való kiterjeszthetőségének köszönhetően. Az XML alapú dokumentumkészítés megvalósítására több szabvány is megjelent, ilyen szabvány például a DITA és a DocBook. A DITA a tartalom orientáltságot és a felhasználói tevékenységet helyezi előnybe. A DocBook egy platform független, különböző rendszerekre jól konvertálható szabvány, ami jól illeszkedik a HTML web böngészőkre. Egyre szélesebb körben való elterjedtsége azt engedi sugallni, hogy a jövőben akár átveheti a LaTeX szerepét. Elterjedését nagy részben elősegíti, hogy a matematikai formulákat is kiválóan meg tudja jeleníteni. A rendszer a MathML szabványt alkalmazza ezen, a matematikai formulákhoz kapcsolódó műveletekre. A MathML számos előnye mellett hátrányokkal is bír. A hosszú, nehezen átlátható, nagy struktúrák azok a hátrányos tulajdonságai, ami miatt a felhasználók kénytelenek valamilyen segédszerkesztő segítségével elkészíteni, s nem kézzel írják meg a MathML kódot. Napjainkban igen szűk a rendelkezésre álló grafikus szerkesztők száma. Ezen okból kifolyólag azt tűztük ki célul, hogy képletek szerkesztésének megkönnyítésére létrehozunk egy egyszerű, emberközeli parancsnyelvet.

A dolgozat témaköre az informatika területén belül egy matematikai formulákat leíró nyelv létrehozatala, az XML alapú DocBook XML nyelvhez. A leíró nyelvünk létrehozásának szükségességét abban láttuk, hogy a nehezen átlátható, hosszú és bonyolult MathML formulák leírását megkönnyítsük, lerövidítsük. Munkánk során teljes mellszélességgel azon voltunk, hogy az eddigi, a miénket megelőző, s ugyanezen a területen működő alkalmazások által szerzett tapasztalatokat a felhasználó minél szélesebb palettán tudja felhasználni. Kutatásunk alatt nemcsak a konvertálás logikájában, hanem a megjelenítésnél is újításokat hoztunk be a korábbi technológiákkal összevetve. A gyorsaság és átláthatóság mellett törekedünk a szebb, a természetesebb megjelenítés létrehozásában. A kutatási munkánk eredménye azoknak lesz elsősorban hasznos, akik informatikai háttérrel rendelkeznek, és webes megjelenítésű dokumentumaikban gyakran alkalmaznak matematikai képleteket. Rendszerünk elkészítésénél szemünk előtt tartottuk a dinamikus bővíthetőség elvét. Elkészült a dolgozat keretében a parancsnyelvnek egy olyan változata, amely az alap szimbolizmust támogatja. Az elvégzett tesztek

alátámasztották a kidolgozott rendszer működőképességét, használhatóságát. A rendszer nem tekinthető lezártnak, hiszen a jövőben új igények léphetnek fel. A kidolgozott rendszer ezen okból is támogatja a deklaratív bővíthetőséget. A dolgozat első részében kitekintést teszünk az eddigi tudományos szövegek szerkesztésére alkalmas eszközök felé, nagyvonalakban bemutatjuk működésüket és összehasonlítjuk ezeket. A formális nyelvek és automaták területére vonatkozólag is készítünk egy rövid összefoglalót, ugyanis kutatásunk további részében szükség lesz a tárgy alapvető fogalmainak ismeretére. Ezután a kidolgozott mintarendszer szerkezetének, működésének fő pontjait mutatjuk be.

A dolgozathoz egy minta rendszer is implementálásra került, mely a csatolt mellékletről telepíthető.

2. ELEKTRONIKUS DOKUMENTUMSZERKESZTÉSI FORMÁTUMOK ÉS MÓDSZEREK ÁTTEKINTÉSE

2.1. LaTeX

A LaTeX [7] egy TeX-en alapuló betűszedő rendszer, amely alkalmas olyan dokumentumok (szakdolgozatok, tudományos cikkek) készítésére, amelyek sok matematikai képletet tartalmaznak. A TeX egy fejlett szövegszedő rendszer, amely nyomdai minőségben állít elő tetszőleges szöveget.

A TeX [11] rendszert Donald Knuth tervezte, készítette, és 1978-ban jelent meg. A tervezés fő szempontja az volt, hogy a rendszer mindenki számára lehetőséget biztosítson, jó minőségű könyvek készítésére, viszonylag kevés erőfeszítéssel, biztosítva, hogy minden számítógépen pontosan ugyanazt a formátumot kapjuk vissza. A TeX rendszer egyik erőssége a matematikai képletek előállítása és kezelése. Ezekért a tulajdonságaiért nagy népszerűségnek örvend tudományos, elsősorban matematikai, számítástechnikai területeken szövegek (cikkek, könyvek) előállításában. A TeX működésének alapelvei [4], hogy rendszerint egy szöveges fájlt szerkesztünk, amely `.tex` kiterjesztéssel rendelkezik. Ez a fájl a dokumentum szövegét tartalmazza, mellette a dokumentum megjelenítésére vonatkozó parancsokat is találunk. Egy TeX parancs rendszerint egy vissza perjellel (backslash) kezdődik, amit követhet egy speciális karakter, vagy tetszőleges karaktersorozat. Az alapvető TeX parancsok alacsony szintűek, azonban léteznek formázási fájlok, csomagok, melyekkel magasabb szintű parancsokat

használhatunk (ezek az alap TeX parancsokat használják), valamint magunk is definiálhatunk makrókat.

Egy TeX-ben írt dokumentum esetén, nem kell törődnünk a szöveg kinézetével, minden szükséges formázást elvégez a program. A parancssorba kiadott a *tex* *fajlunkneve.tex* utasítás segítségével előáll a forrásfájlból egy naplófájl és egy DVI fájl. Ez egy vektorgrafikus dokumentum formátum. Itt már azt láthatjuk, ami a nyomtatás után meg fog jelenni a papíron.

A rendszer része egy viszonylag egyszerű szövegjelölő nyelv, amely alapján a szöveget először ellátjuk a megjelenítési információkat hordozó utasításokkal. A jelöléssel ellátott szövegből azután egy fordítóprogram létrehozza a megjeleníthető dokumentumot.

A LaTeX [13] ma a legnépszerűbb dokumentumszerkesztők egyike. Számos irodalom foglalkozik használhatóságával, kezelésének útmutatójával. Előnyei közé tartozik, hogy számos tipográfiai feladat megoldására képes, használható nyomdai minőségű kimenet előállítására. A szerzőnek elég a dokumentum logikai struktúráját leírnia, s emellett nem kell foglalkoznia a megjelenítéssel, ugyanakkor lehetséges a megjelenés teljes mértékű szabályozása. Könnyen lehet tartalomjegyzéket, irodalomjegyzéket és lábjegyzeteket generálni a felhasználónak. A LaTeX jól kezeli a képleteket és folyamatosan fejlődik [6]. Az előnyei közé sorolandó még, hogy hordozható, kevés memóriát igényel, ingyenes, valamint szabad forráskódú (korlátozások csak a márkanevek használatában és az üzleti célú felhasználásban vannak). Az előnyök mellett néhány hátrány is megjelenik a rendszerben. A LaTeX nehezen áttekinthető, s bizonyos feladatokat nehezebb vele megoldani, mint egy WYSIWYG rendszerben. Külön leíró nyelvet használ, amit nem mindig egyszerű megérteni. Ennek megismeréséhez nem mindig áll rendelkezésre elég dokumentáció. Az alkalmazás használata során észrevesszük, hogy a hibák kijavítása nehézkes, nehéz új kinézeti tervet készíteni.

2.2. WYSIWYG szerkesztők

A WYSIWYG [15] szerkesztők alapelve, hogy a felhasználó már a szerkesztési munka alatt láthatja, közvetlenül ellenőrizheti a kimeneti formátumot. A piacon számos grafikus szerkesztővel találkozhatunk: Dreamweaver, Microsoft FrontPage, NetObjects Fusion, OpenOffice.org, WebEditor, Netscape. A WYSIWYG szerkesztők erősen a kényelmes olvasási látványhoz vannak optimalizálva. A szerkesztőik grafikus menürendszert

biztosítanak a felhasználó számára, egyszerűsítve ezzel a parancskódolást, vagy a szerkesztési lépéseket. Elég egyszerűen „kattintgatni” az egérrel az eszköztáron, s a formázás máris készen áll. Hátránya, hogy távoli gépen lévő dokumentumok szerkesztése elég nehézkes vele, valamint rengeteg számolással jár egy-egy utasítás kiadása (pl.: hova akarunk tenni egy képletet). Elmondható az is, hogy sok esetben a WYSIWYG elfedi a forráskódban meglévő tartalmi hibát, még más fontos esetekben a felhasználói oldal rugalmas megjelenítési igényeinek nem tesz eleget.

2.3. DocBook

A DocBook [9][10] egy szemantikai jelölőnyelv, mely lehetővé teszi a felhasználók számára, hogy a dokumentum tartalmát bemutattassa semleges formában, ami jelöli a logikai szerkezetet, s a tartalmat. DocBook képes HTML, XHTML, PDF formátumban is megjeleníteni a tartalmat, anélkül, hogy a felhasználók számára bármilyen változást kellene alkalmazni a forrásfájlon. Az új DocBook verziókba be lehet ágyazni matematikai képleteket is, amihez a szabvány MathML formulákat használ. A DocBook az XML nyelvet használja. A jelenlegi DocBook verzió nyelve hivatalosan RELAX NG sémára illetve integrált Schematron szabályokra illeszkedik.

A DocBook [5] dokumentumok elsődlegesen nem azt írják le, hogy a tartalom hogy nézzen ki, hanem magát a tartalmat adják meg. Az absztrakt leírást egy külső feldolgozó segítségével alakíthatjuk át vizuálisan formázottá, így előállítva a kimenetet. A DocBook az XML szabványra épül, a szemantikai elemekhez úgynevezett címkéket rendel. Ezeket három nagy kategóriába lehet sorolni: *strukturális*, *blokk-szintű* és *sorszintű* (inline). Strukturális címkék adják széleskörű jellemzőket, azok tartalmát, például meghatározza, hogy melyek a gyerek elemek. Blokk-szintű elemek például a bekezdés, listák stb.. Nem minden aktuális szövegrész tartalmaz közvetlenül ilyen elemet. A szekvenciális blokk szintű elemek megmutatják a következő lépés irányát. Az inline szintű címkéjű elemek alkotják például a hivatkozásokat. Ehhez kapcsolódóan kezelhetjük a tipográfiai elemeket, betűtípus kijelölést, az ezekhez kapcsolódó attribútumokat. A hitelesítés (validálás) egy előre meghatározott DocBook séma alapján történik. Egy XML dokumentum érvényesítése megfelelő programozási eszközök (Validator) segítségével lehetségesek.

A DocBookban XML dokumentumok hozhatók létre és szerkeszthetők bármilyen szövegszerkesztővel. Egy dedikált XML szerkesztő egyben funkcionális DocBook szerkesztő.

DocBook fájlok készítésénél használt kimeneti fájlok a legkülönbözőbb formákban szerepelhetnek. Az XSLT stíluslapokkal a DocBook dokumentumokat több formátumban (HTML, XSL-FO későbbi átalakítás PDF, stb.) szerkeszthetjük. A DocBook dokumentumok konverziója, rendszerint az XSL-FO szabvánnyal kerül végrehajtásra. Az XSL-FO stíluslapok elég kifinomultak ahhoz, hogy létrehozzák a tartalomjegyzékeket, a szövszedeteket, s az indexeket. Mivel a szabvány DocBook XSL stíluslapok jól formált XSL stíluslapok, és a DocBook jól formált XML, a felhasználók maguk is készíthetnek egyéni stíluslapokat vagy akár egy teljes értékű programot, hogy feldolgozzon egy DocBook állományt és egy megfelelő kimeneti formátumra, igényeknek megfelelően átkonvertálja.

A jelenlegi verzió a DocBook, 5.0. Korábbi verziókat még ma is széles körben használják. A DocBook 4.x dokumentumok nem kompatibilisek a DocBook 5-el, de konvertálhatóak DocBook 5-re. A DocBook számos funkciót tartalmaz, melynek elsajátítása viszonylag rövid időt követel.

2.4. Összefoglalás

A következőkben röviden összefoglaljuk az elterjedtebb dokumentumszerkesztő módszereket a funkcionalitásuk értékelése alapján.

Összehasonlító táblázat

1. táblázat

Szempont	LaTeX	XML, DocBook	WYSIWYG
Prezentáció	tartalom együtt a struktúrát jelentő parancsokkal	tartalom és struktúra szoros kapcsolatban	alkalmazástól függ
Szerkesztés	text editorban, kódolás	text editorban, alkalmas XML szerkesztőben	desktop alkalmazás segítségével
Méret	kis méret	terjengős, nagyobb méret	alkalmazástól függ

Szempont	LaTeX	XML, DocBook	WYSIWYG
Kimenetek	különböző kimenetek generálhatóak (dvi, pdf, html)	különböző kimenetek készíthetők (pdf, html)	kimenetek típusa alkalmazástól függ
Terjeszthetőség az interneten	kevésbé jól	nagyon jól (web böngészők támogatása)	általában nem jól, a minőség jelentősen romlik
Javíthatóság	bonyolult kód, nehezen átlátható	hosszú, bonyolult kód, nehezen átlátható	nem mindig az történik, amit szeretnénk
Támogatottság	számtalan egyedi csomag elérhető	új szabvány, még kevésbé jó a támogatottság	a gyártótól függ
Bővíthetőség	egyszerűen bővíthető	jól bővíthető	általában nem bővíthető
Matematikai képletek	könnyű megtanulni az alapokat, de vannak nagyon bonyolultak is	MathML: alapok könnyen elsajátíthatóak, de terjengő a kód	eszköztárak segítségével, nehézséges

Szempont	LaTeX	DocBook	WYSIWYG
Példa: $a^2 + b^2 = c^2$	$a^2 + b^2 = c^2$	<pre> <math> <mrow> <mrow> <msup> <mi>a</mi> <mn>2</mn> </msup> <mo>+</mo> <msup> <mi>b</mi> <mn>2</mn> </msup> </mrow> <mo>=</mo> <msup> <mi>c</mi> <mn>2</mn> </msup> </mrow> </math> </pre>	képletszerkesztő eszköztár segítségével, nagyjából 6-7 kattintás

3. ELEKTRONIKUS DOKUMENTUMSZERKESZTŐK ÁTTEKINTÉSE

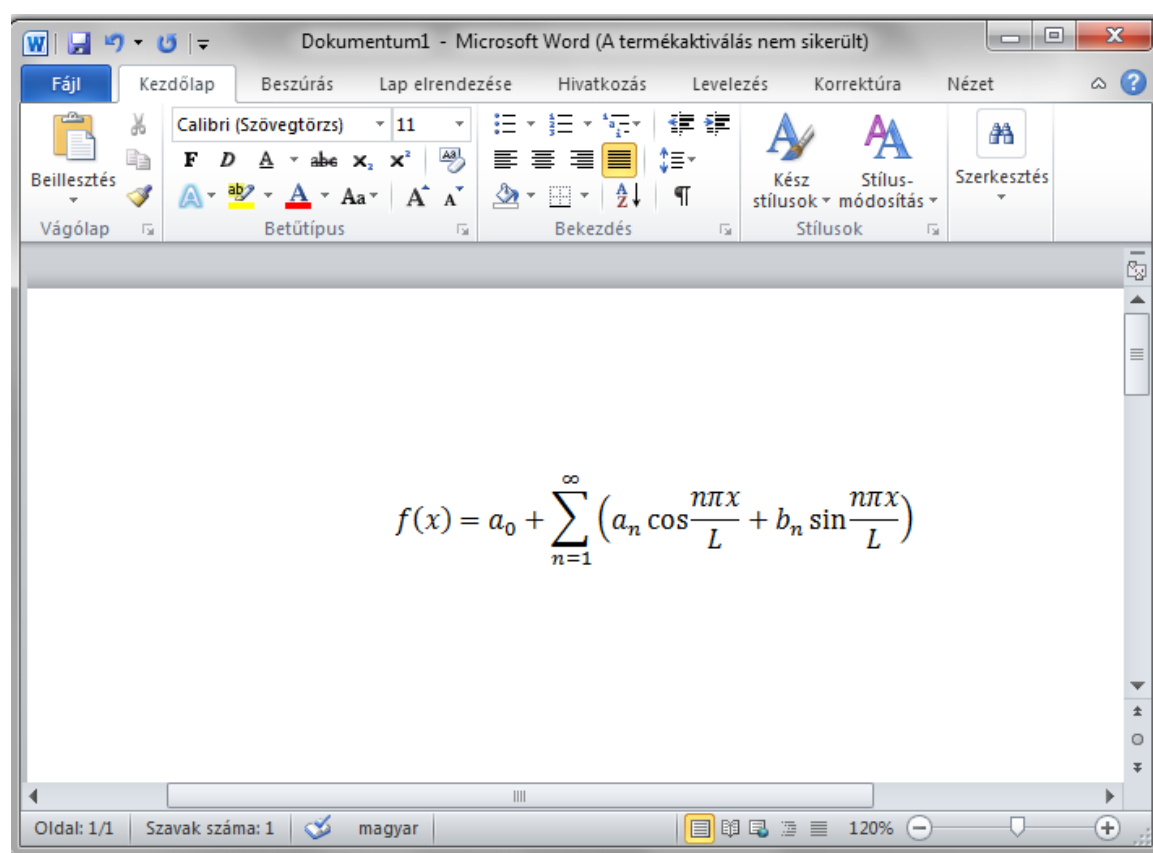
3.1. Microsoft Word

A grafikus megjelenítési módot alkalmazó szerkesztők közül a Word volt az első népszerű szövegszerkesztő [14]. Szerkesztés közben a képernyőn való megjelenítéséhez, olyan lehetőségeket kínál, mint például a félkövér, dőlt vagy az aláhúzott. Az őt megelőző szövegszerkesztők (WordStar és a WordPerfect) egyszerű, csak szöveges megjelenítést alkalmaztak, a képernyőn jelölőkódokkal, vagy néha különböző színekkel. A különböző speciális szerkesztések a menükben, vagy az eszköztárakban érhetők el. A szerkesztő mostani verziói többet tudtak, mint csupán a szövegszerkesztés. A Rajz eszköztár segítségével egyszerű ábrákat rajzolhatunk dokumentumainkba, otthoni szinten.

Professzionális kiadványszerkesztők sokkal alkalmasabbak ilyen célú feladatokra. Lehetőségünk van egyszerű diagramok, gráfok készítésére is.

A Microsoft Word a legelterjedtebb szövegszerkesztő, ezáltal formátuma (DOC) szinte már szabvánnyá vált.

A mostani Word szerkesztők már matematikai képleteket meglepően jól tudnak kezelni, de számos hiányossággal is küzdenek, minthogy a képletek megjelenítésébe nem szólhatunk bele, kötött a szerkesztő, a konvertálás során a képletek minősége romlik és az interneten nem terjeszthető jól.



1. ábra A Microsoft Word 2010 felhasználói felülete

3.2. Oxygen

Az Oxygen egy platform független, univerzális Java alkalmazás, mely XML alapú dokumentumok szerkesztéséhez számos funkcióval bír. Egy dokumentumérvényesítéshez több séma is létezik az alkalmazásban, mint a DTD, XML Schema, Relax NG, NRL és NVDL sémák.

Az Oxygen XML három nézeti szerkesztési lehetőséget kínál: a szöveges, a rács és a szerző nézet. A szöveg nézet, ahogy a neve is mutatja, a dokumentum szöveges részét szerkeszti. Nincs szabványos mechanizmus a séma formátumok dokumentumhoz való kötéséhez. Az alkalmazás egy feldolgozó utasítás segítségével választja ki a megfelelő séma használatát. A rács nézet, az XML dokumentumot egy táblázatkezelő- szerű módon szerkeszti. A bal szélső oszlopban a gyökérelemet (ami tartalmazza a megjegyzéseket és a feldolgozási utasításokat is) tartalmazza. A következő oszlopban a gyökérelem attribútumait. A rácsnézet XML formátumok számára hasznos, melyek táblázatokra hasonlítanak. A szerző nézet az Oxygen 9-es verziójában jelenik meg, ezzel egy WYSIWYM felületet nyújtva e dokumentumok szerkesztéséhez. A megjelenést egy CSS segítségével befolyásolhatjuk. Az Oxygen a DITA, DocBook és a TEI CSS formátumokat támogatja.

3.3. XMLSpy

Az XMLSpy [12] egy olyan XML fejlesztő, mely lehetővé teszi a fejlesztők XML alapú webes szolgáltatások és alkalmazási technológiák létrehozásában. Ilyen például: XML, XML- Schema, XSLT, XPath, XQuery, WSDL és SOAP. Az alkalmazás elérhető Microsoft Visual Studio és Eclipse plugin alatt is.

Az XMLSpy számos nézeti és szerkesztési funkciót tartalmaz. Az XML dokumentumok létrehozása és szerkesztése mellett lehet vizuális XML sémát vagy DTD-t szerkeszteni. Mindezek mellett a webes szolgáltatások, grafikus WSDL létrehozása és szerkesztése, SOAP, XQuery fejlesztés és hibakeresés, valamint XPath fejlesztés és elemzés gazdagítja tulajdonságait. Előnye, hogy egyszerűen használható, valamint számos és fejlett fejlesztési szolgáltatásokkal rendelkezik.

3.4. Integre MathML Equotation Editor

Az *Integre MathML Equotation Editor* (projekt név: *Zed*) egy grafikus MathML képletszerkesztő.

Működése egyszerű: menük vagy paletták segítségével kiválasztjuk a nekünk megfelelő primitívet, és ezt a szerkesztő segítségével kitöltjük, részben gépeléssel (számok, változók), részben újabb primitívekkel.

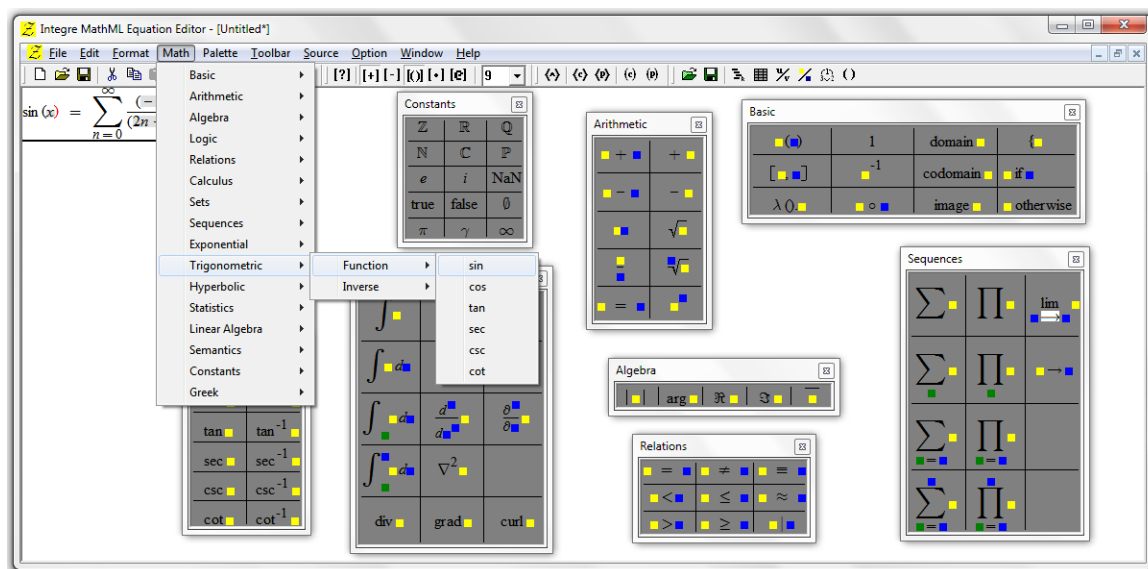
A probléma az, hogy a MathML komplexitása miatt a primitívek száma is nagy lett, hogy kielégítsen minden felhasználói igényt. Ennek egyenes következménye, hogy a

felhasználó hosszú percekig keresi a neki megfelelő menüpontot, mire megtalálja, amire szüksége van. A programban lehetőségünk van paletták használatára, de itt hasonló nehézségbe ütközünk, mint az előbb: a paletták sok helyet foglalnak el a képernyőből, és a sok ábra is összezavarhatja a felhasználót (bár ilyen módon valószínűleg hamarabb megtalálja, amit keres).

Amennyiben találkozunk olyan feladattal, amit nem tudunk sem a menük, sem a paletták segítségével megoldani, közvetlenül szerkeszthetjük a MathML kódot, ami viszont az ajánlás referenciájának ismeretét követeli meg.

Az 1. ábra az Integre alkalmazás grafikus felhasználói felületét mutatja. Hátránya látható: hiába a jól felépített menük és paletták, egy primitív megtalálása így sem egyszerű, valamint a szerkesztő rendelkezik bizonyos hiányosságokkal (például: felső indexben szereplő kifejezéseket zárójeleznünk kell, nem oda kerül leütött karakter, ahol a kurzor áll), ami miatt a szerkesztés bonyolultabb lesz.

Belső működését tekintve az Integre a képleteket saját formalizmusain (Zed-C és Zed-P) építi fel, tárolja, majd ezt konvertálja MathML (Presentation vagy Content Markup) szabványra. Előnyére válik, hogy ennek a nyelvnek a segítségével magát az alkalmazás menüit, palettáit testre tudjuk szabni, ezáltal bővítve az alap funkciókat.



2. ábra Az Integre grafikus felhasználói felülete

4. ÁTTEKINTÉS A REGULÁRIS NYELVEKRŐL

A programozási nyelvek elméleti alapját szabályokon alapuló formális nyelvek, környezetfüggetlen nyelvek (Chomsky-féle 2. nyelv osztályba tartozó nyelvek) adják. Mivel az általunk létrehozott nyelv is formális nyelv, ezért szükséges egy kis elméleti kitekintést tennünk. [1]

4.1. Formális nyelvek alapjai

Ábécének nevezzük a tetszőleges jelek véges halmazát (jelöljük: Σ). Az ábécé elemeit *betűnek* nevezzük. Az ábécé betűiből *szavakat* (jele: w) képezhetünk, a betűk egymás után írásával, egy szó *hosszán* (jele: $|w|$) az azt alkotó betűk számát értjük. Az üres szó olyan szó, amelynek a hossza 0 (jele: ε).

Jelölje Σ^* az ábécé feletti összes lehetséges szavak halmazát, beleértve az üres szót is ($\Sigma^* = \{a_1 a_2 \dots a_n | a_1, a_2, \dots, a_n \in \Sigma, n \geq 0\}$), a Σ^+ az ábécé összes lehetséges nem üres szavának halmazát ($\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$), Σ^i pedig a pontosan i darab betűből álló szavak halmazát ($\Sigma^0 = \{\varepsilon\}$).

Bevezetünk egy bináris műveletet is Σ^* fölött, még hozzá két szó *konkatenációját* (összefűzését, sorozatát). Legyen $u, v \in \Sigma^*$, $u = a_1 a_2 \dots a_n$ és $v = b_1 b_2 \dots b_m$, u és v konkatenáltján az $uv = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$ szót értjük (nyilvánvaló: $|uv| = |u| + |v|$). a konkatenáció művelete asszociatív, nem kommutatív, és van egységeleme ($u\varepsilon = \varepsilon u = u, \forall u \in \Sigma^*$), tehát Σ^* a konkatenációval monoidot (egységelemes félcsoporthoz) képez. Azt mondjuk, hogy v *kezdőszelete* (*prefixe*) az u szónak, ha létezik z szó úgy, hogy $u = vz$, ha $z \neq \varepsilon$, akkor u valódi kezdőszelete u -nak. Egy szó *végszeletét* (*suffixét*) analóg módon definiálhatjuk. A v szó *résszava* u szónak, ha $\exists p$ és q szavak úgy, hogy $u = pvq$. Ha $pq \neq \varepsilon$, akkor v *valódi résszava* u -nak. Az $u = a_1 a_2 \dots a_n$ szó *tükröképén* az $u^R = a_n a_{n-1} \dots a_1$ szót értjük.

A Σ^* tetszőleges L részhalmazát Σ ábécé feletti *nyelvnek* nevezzük. Különbséget teszünk az $L = \emptyset$ üres nyelv és az $L = \{\varepsilon\}$ üres szóból álló nyelv között. Egy $w = a_1 a_2 \dots a_n \in \Sigma^*$ jelsorozat, ahol $a_1, a_2, \dots, a_n \in L$, L *mondatának* nevezzük.

Ha L, L_1, L_2 mind Σ fölötti nyelvek, akkor értelmezzük az *egyesítést* ($L_1 \cup L_2 = \{u \in \Sigma^* | u \in L_1 \text{ vagy } u \in L_2\}$), *metszet* ($L_1 \cap L_2 = \{u \in \Sigma^* | u \in L_1 \text{ és } u \in L_2\}$), *különbség* ($L_1 \setminus L_2 = \{u \in \Sigma^* | u \in L_1 \text{ vagy } u \notin L_2\}$), *komplementum* ($\bar{L} = \Sigma^* \setminus L$), *szorzat* ($L_1 L_2 =$

$\{uv \in \Sigma^* | u \in L_1, v \in L_2\}$), *hatvány* ($L^0 = \{\varepsilon\}, L^n = L^{n-1}L$, ha $n \geq 0$) és *tükrözés* ($L^R = \{u^R | u \in L\}$) műveletét.

A nyelveket megadhatjuk elemeinek felsorolásával ($L = \{\varepsilon, ab, aabb, aaabbb, \dots\}$), tulajdonság segítségével ($L = \{a^n b^n | n = 0, 1, 2, \dots\}$) vagy nyelvtannal. Az utóbbi magyarázatra szorul.

Nyelvtannak nevezünk minden olyan $G = (N, T, P, S)$ rendezett négyest, ahol N a *változók* ábécéje, T a *terminális jelek* ábécéje ($N \cap T = \emptyset$), P a *produkciós (helyettesítési) szabályok* véges halmaza ($P \subseteq (NUT)^* N (NUT)^* \times (NUT)^*$), $S \in N$ a nyelvtan *kezdőszimbóluma* (*mondatszimbóluma*). A produkciós szabályok leírására az $u \rightarrow v$ jelölést használjuk, ahol u -t a szabály bal oldalának, v -t a szabály jobb oldalának nevezzük. Ha több olyan szabályunk is van, amelyek bal oldala megegyezik, azt egyszerűen így írhatjuk: $u \rightarrow v_1 | v_2 | \dots | v_n$.

Mondatszerű formának olyan jelsorozatokot nevezünk, amiben nem csak az ábécé jelei, hanem nyelvtani szimbólumok is szerepelhetnek. Vezessük be a *közvetlen levezetés* ($\Rightarrow$) relációt. Legyenek u, v mondatszerű formulák. Ekkor:

$$u \Rightarrow v, \quad \text{ha } u = p_1 p p_2, \quad v = p_1 q p_2, \text{ és } (p, q) \in P.$$

Ez annyit jelent, ha van u , mely tartalmaz olyan p részszót, amelyet q -ra cserélve a produkciós szabályok alapján v -t kapjuk.

A közvetlen levezetés reláció tranzitív és reflexív lezártját $\Rightarrow^*$ jellel jelöljük, ebből következik, hogy $u \Rightarrow^* v$, ha léteznek $w_0, w_1, \dots, w_n \in (N \cup T)^*, n \geq 0$ szavak és $u = w_0, w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \dots, w_{n-1} \Rightarrow w_n, w_n = v$. Ha ki akarjuk hangsúlyozni, hogy a levezetés mely nyelvtan szabályaival történt, akkor a nyíl alá írjuk a nyelv jelét $\Rightarrow_G^*$ (a szövegszerkesztő korlátai miatt előfordul, hogy ebben a dolgozatban alsó indexben tüntetjük fel).

Egy adott G nyelvtan által generált L nyelv jelölése $L(G)$, és a nyelv Σ^* azon elemeit tartalmazza, melyek a G nyelvtan segítségével tetszőleges számú lépésben generálhatóak:

$$L(G) = \{w \in \Sigma^* | S \Rightarrow_G^* w\}$$

Szükséges továbbá megemlíteni néhány jelölési konvenciót: a nem terminális szimbólumokat a latin ábécé elejéről, nagy betűkkel jelöljük (A, B, C), a terminális

szimbólumokat a latin ábécé elejéről vett kisbetűkkel (a, b, c), a terminális szimbólumokból álló jelsorozatok a latin ábécé végéről vett kisbetűkkel (x, y, z), a mind terminális, mind nem terminális szimbólumokat tartalmazó mondatszerű formákat görög kisbetűkkel (α, β, γ) jelöljük.

A nyelveket tehát jellemezhetjük nyelvtanok segítségével. A nyelvtanokat meghatározó produkciós szabályok bonyolultsága alapján, Chomsky a nyelveket négy osztályba sorolta. Minél magasabb nyelvosztályba tartozik egy nyelv, az őt előállító nyelvtan produkciós szabályainak egyre szigorúbb feltételeknek kell eleget tenniük.

Chomsky-féle nyelvosztályok:

- *0-ás nyelvosztály (rekurzívan felsorolható nyelvek):* semmilyen megkötés nincs a produkciós szabályokra. Minden olyan nyelvet előállítanak, amelyeket a Turing-gép elfogad.
- *1-es nyelvosztály (környezetfüggő):* a nyelvet előállító G nyelvtan minden produkciós szabálya $\beta A \gamma \rightarrow \beta \alpha \gamma$, ami annyit jelent, hogy az $A \rightarrow \alpha$ produkciós szabály csakis $\beta - \gamma$ környezetben alkalmazható. Ilyenek a nondeterminisztikus Turing-automata által elfogadott nyelvek.
- *2-es nyelvosztály (környezetfüggetlen):* egy nyelvet előállító G nyelvtan minden produkciós szabálya $A \rightarrow \alpha$ alakú. Interpretációja annyit tesz, hogy az A nem terminális szimbólumot mindig helyettesíthetjük az α jelsorozattal, függetlenül a környezetétől. Nondeterminisztikus veremautomaták által elfogadott nyelvek osztálya, a programozási nyelvek elméleti hátterét adja.
- *3-as nyelvosztály (reguláris nyelvek):* a nyelvet előállító G nyelvtan minden produkciós szabály $A \rightarrow a$ illetve $A \rightarrow aB$ alakú, tehát a szabályok bal oldalán mindig egy nemterminális szimbólum áll, jobb oldalán pedig egy terminális, vagy egy terminális és egy nemterminális szimbólum áll. Engedélyezett az $S \rightarrow \varepsilon$ alakú szabály is, amennyiben S nem áll egyik szabály jobb oldalán sem. Ezek a nyelvtanok generálják a reguláris nyelveket.

5. MATEMATIKAI KÉPLETSZERKESZTÉS ESZKÖZEINEK ÁTTEKINTÉSE

5.1. A LaTeX formalizmusa

A TeX [2][4] nyelv egyik előnye, hogy szép matematikai képleteket készíthetünk vele dokumentumaink számára. Matematikai kifejezések írására úgynevezett matematikai környezetben van lehetőségünk. Ezt a forrás fájlban \$ jelek vagy \[\] közé írt betűk, számok és TeX parancsok kombinációja. A TeX két speciális matematikai környezetet is megkülönböztet, név szerint a sorközi módot és a kiemelt (blokk) módot.

A sorközi mód kódját \$ jelek közé írjuk, a kiemeltet pedig \$\$ jelek közé. A jelölés rendszer könnyen megtanulható, de a formalizmus miatt a bonyolult képleteket nehéz áttekinteni.

Matematikai módban a speciális tipográfiai formázás, a betűk közti távolság, operátorok körüli távolság, karakterméretek mind módosulnak. A betűk kurzív formázást kapnak.

A képletek leírása meglehetősen tömör [3]. Például hatvány, felső index parancsa a ^, az alsó indexé pedig _, tehát a x_i^2 képletet TeX szövegben a `$x^2_i$` kód adja meg. Ezen a példán is látszik, hogy ha hosszabb formulát akarunk tipográfiaileg hasonlóan formázni (például felső indexbe tenni), akkor csoportokat kell készíteni. A csoport tagjait { } közé zárjuk (tehát: x^{a+2b+c} kódja `$x^{a+2b+c}$`). Nagy operátorokat, görög betűket TeX parancsokkal tudunk a képletbe írni (például az $\int$ parancsa az `\int`, a β a `\beta`, az Ω az `\omega`).

Ezeket felül TeX [6] parancsokkal megadhatunk képletszámozást is, formázhatjuk a karakterek méretét. Különböző rendezett szerkezeteket is meg tudunk adni, de a TeX nyelv parancsai kicsit nehézkesek ekkor (például mátrix), de léteznek csomagok, amik megkönnyítik ezek megadását. Ilyen parancsokat, makrókat a LaTeX is tartalmaz, valamint egyik csomagja, az *amslatex* (American Mathematical Association saját matematikai formázásokat tartalmazó csomagja) is sok leírást tartalmaz a formulák megadásához.

5.2. MathML nyelv

A *Mathematical Markup Language* (MathML) [8], egy XML alapú, matematikai formulákat leíró nyelv. Célja a matematikai formalizmust szabványos módon beágyazni a világhálón található dokumentumokba. A MathML szabványnak két fajtája létezik: a megjelenítést leíró (Presentation Markup) és a tartalmat leíró (Content Markup)

MathML nyelv minden szolgáltatásának lefedésére. Mivel az új nyelvet a DocBook környezetben kívánjuk használni, emiatt néhány további egyszerűsítési lehetőséggel is élhetünk.

Stílusok, formázások megadását kihagytuk, mivel ez általában a kész képlet kisebb változtatásait foglalja magában, azért, hogy a kimeneten a képlet jól nézzen ki, összhangban legyen környezetével. Egy-egy stílusjegy megváltoztatása (például: egy rész mérete) csak egy-egy tag (tagpár) beszúrását jelenti, ami a leíró nyelv többszöri (felesleges) újrafordítását eredményezné.

A kidolgozott *prefix nyelv* egy speciális reguláris nyelv, mely formulák megadására szolgál, s felhasználók számára készült. A LaTeX nyelvénél egyszerűbb, a leggyakoribb funkciók tömörek, könnyen értelmezhetőek.

A nyelv alapvető tulajdonsága, hogy egy matematikai kifejezésből (gondoljuk először valami rövide, például: $a+b$) kiemeljük a tagokat és kiemeljük az operátort, majd a tagokat zárójelben, vesszővel elválasztva egymás mögé írjuk, a zárójel elé pedig odaírjuk az operátort (az operátor nevét, így: $plus(a,b)$). Ilyen módon egyfajta függvényszerű (inkább a programozási nyelvekből ismert függvényre, mintsem matematikai értelmezésben ismert függvényre kell asszociálni) alakot kapunk, ahol a függvény neve (előtagja, zárójel előtti része) kijelöli a műveletet, a zárójelben pedig a tagok, mint a függvény paraméterei, vesszővel követik egymást. A nyelv megenged tetszőleges egymásba ágyazódást annak érdekében, hogy bármely képlet felírható legyen ily módon.

A prefix alak aritmetikai, logikai műveleteknél lehet, kicsit abszurdnak tűnik, és bonyolítja a felépítését egy prefix formulának, de egyszerűsítési, kényelmi funkciókat az alkalmazás fog biztosítani a szerkesztéshez a felhasználó számára. Ez az alak viszont kedvez a nagyoperátorokkal jelölt műveleteknek, mert olvasási irányban kell az operátorhoz kapcsolódó tagokat beleírni a prefix alak paraméter listájába.

$\int_a^b 2x \, dx.$	
Integrál a -tól b -ig kétszer x az x szerint	
Prefix	LaTeX
<code>int(a,b,*(2,x),x)</code>	<code>\int_a^b \! 2x \, \, \mathrm{d} x.</code>

4. ábra Prefix formula viszonya a beszélt nyelvhez és a LaTeX-hez

6.2. A prefix nyelv szabályai

A bemutatott elméleti alapokat megismerve, most áttekintjük a prefix nyelv szabályrendszerét. A prefix nyelvet egy olyan

$$G_1 = (N, \Sigma, P, S)$$

nyelvtan generálja, ahol

$$N = \{S, L, F, A, P, V, N\}$$

$$\Sigma = \{a, b, \dots, z, A, B, \dots, Z, 0, 1, \dots, 9\}.$$

A produkciós szabályokat a 5. ábra mutatja.

$$\begin{array}{lcl} S & \rightarrow & L \mid N \\ L & \rightarrow & F \mid V \\ F & \rightarrow & P(A \\ A & \rightarrow & S, A \mid S) \\ P & \rightarrow & [a..z]^+ [0..9]^* \\ V & \rightarrow & [a..z] \mid [A..Z] \\ N & \rightarrow & [0..9]^* \end{array}$$

5. ábra A prefix nyelvet generáló nyelvtan produkciós szabályai

A szabályok előző leírásának célja a megértés segítése. Láthatóan nem szabályos a leírás, tartalmaz egyéb karaktereket, mint terminális karaktereket (ilyenek a: vessző és a zárójelek) valamint tartalmaz *reguláris kifejezéseket* (egyfajta lokális ábécéket, például: $[a..z]$ jelölés az ábécé kisbetűiből pontosan 1 darab, + esetén egy vagy több, * esetén 0 vagy több). A nem terminális szimbólumok megválasztása utal az általuk betöltött szerepre, S a *start* szimbólumot jelöli, L az angol *letter* szóra utal, N a *number*, F a *function*, V a *variable*, P a *prefix*, A az *attribute* angol szavakra utalnak. Az is látszik, hogy az N szabályban indirekt benne van az $N \rightarrow \varepsilon$ szabály is. Hangsúlyoznánk, hogy a nyelvtan csupán szemléltetésre jó.

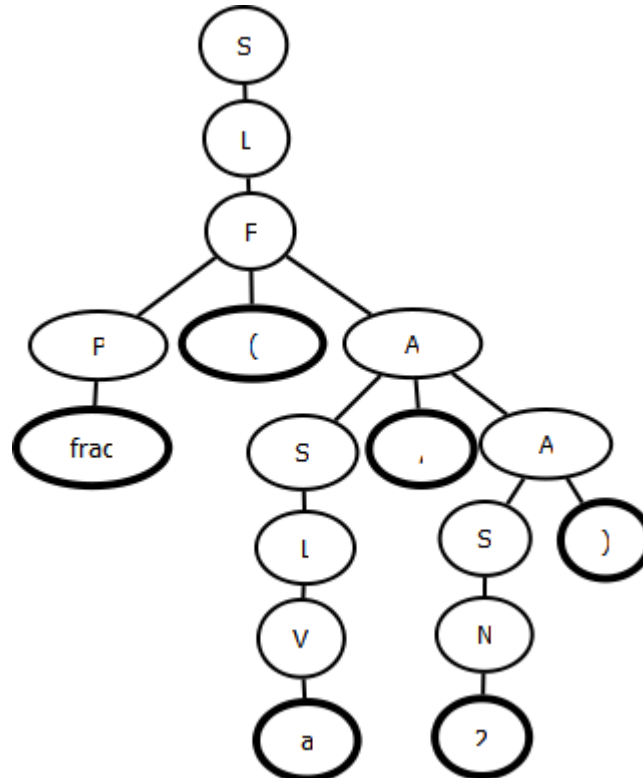
A prefix nyelv alapelemei a terminális elemek, melyeken a prefix függvény által jelölt műveletek értelmezve vannak. Ezen elemnek két fajtája van: tetszőleges hosszúságú szám, vagy egy darab betű (változó). A prefix függvény a nyelv egy olyan eleme, amely egy karakterrel kezdődik, majd további karakterekkel folytatódhat, amiket számok követhetnek. Ezután egy nyitózárrójel következik, amit tetszőleges prefix nyelvbeli elem követ (akár terminális, akár függvény), akár több is, ilyenkor vesszővel elválasztva, majd egy záró zárrójel zárja az egészet.

A G_1 nyelvtan könnyen belátható, hogy *környezetfüggetlen* nyelvtan.

Nézzünk egy példát a levezetésre: a prefix nyelvvel az $\frac{a}{2}$ -t $\text{frac}(a,2)$ formulával írhatjuk le. Ennek egy levezetését mutatjuk be alább:

$$S \rightarrow L \rightarrow F \rightarrow P(A \rightarrow \text{frac}(A \rightarrow \text{frac}(S, A \rightarrow \text{frac}(V, A \rightarrow \text{frac}(a, A \rightarrow \text{frac}(a, S) \rightarrow \text{frac}(a, N) \rightarrow \text{frac}(a, 2)$$

6. ábra Egy egyszerű példa, levezetés



7. ábra A példa levezetési fája

Látható, hogy a behelyettesítéseket más sorrendben elvégezve is erre a formára juthatunk. Egy bonyolultabb példán keresztül azt is bemutatjuk, hogy a nyelv egyértelmű. Tekintsük az $a^2 + b^2 = c^2$ formulát. Prefix nyelven ezt így írjuk le:

$$\text{eq}(\text{plus}(\text{pow}(a, 2), \text{pow}(b, 2)), \text{pow}(c, 2))$$

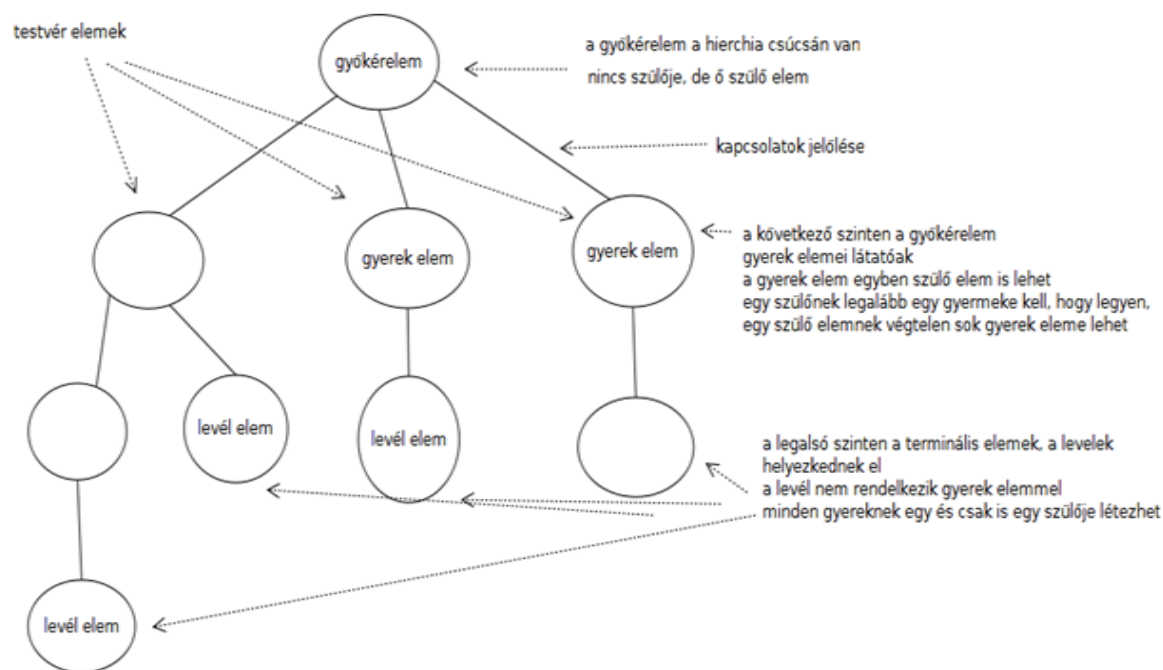
Az átalakítás levezetését terjedelme miatt a mellékletek közé helyeztük el (1. sz. melléklet)

7. ÁTALAKÍTÁSI ELVEK

7.1. Prefix fa

A prefix nyelv egyszerű számítógépi kezelésére létrehoztunk egy logikai adatszerkezetet, a *prefix fát*, amiben egy prefix formulát tárolhatunk. A prefix fa úgy épül fel, hogy biztosítsa azt, hogy a benne tárolt prefix formula jól formált formula legyen.

A prefix fa egy gyökeres fa, tehát egyetlen egy gyökér elemmel rendelkezik. A gyökérelem a fa hierarchiájában a legfelső szinten áll, melynek nem létezik szülő eleme, gyereke eleme lehet 0 vagy végtelen sok (megjegyzés: természetesen az implementáció során van fizikai korlátunk a gyerekek maximális számára). Minden csomópontnak maximálisan egy szülője lehet és nulla vagy egy jobbtestvére. A levélnek nem létezik gyerek eleme. A leveleken mindig terminális elemek állnak.



8. ábra A Prefix fa felépítése

7.2. Átalakítási alapelvek

A következőkben lépésről-lépésre vesszük az egymást követő logikai szintek közötti átalakítások elveit.

7.2.1. Matematikai képlet átalakítása prefix formulává

A matematikai képletből prefix formulává való konvertálásának elsődleges alapelve a precedencia sorrend vizsgálata, mely a matematikában használt precedencia sorrendiség

elvét követi. A képletben lévő legalacsonyabb precedencia szinten lévő elem kerül a formula legelejére. Mind a matematikai operátorokból, mind a matematikai függvényekből prefix függvényeket képzünk, olyan módon, hogy ezekből beszélő neveket hozunk létre, amelyek a prefix függvény fejrészét alkotják. Az általuk érintett vagy összekapcsolt matematikai kifejezéseket a prefix függvény paraméter listájában adjuk meg zárójelpárok között, vesszővel elválasztva egymástól. A kifejezések egymásba ágyazhatóak. A változókból és a konstansokból terminális elemeket képzünk.

Következzen egy egyszerű példa. Tekintsük a következő matematikai képletet:

$$a^2 + b^2 = c^2$$

Prefix formává alakítás során először a legkisebb precedenciájú elem értékelődik ki. Ez az egyenlőség jel, amit leírjuk és egy zárójelpárral ellátjuk, =(,). Az operátornak két oldala van: a bal és a jobb, ezért a zárójelpáron belül két tagunk lesz, amit vesszővel választunk el. A kiértékelésnél a bal oldalt vizsgáljuk. Itt a + operátor a legkisebb precedenciájú elem, aminek szintén jelen esetben két gyerek eleme lesz, ezért, amikor megkapja a zárójelpárt, közte neki is két tagot jelölünk meg vesszővel elválasztva =(+(,)). Az a^2 és a b^2 ugyanazon az oldalon egyforma precedencia szintet képvisel, így bal-jobb haladást követünk. A példánkban látható, hogy a hatvány függvényt pow függvénynévvel láttuk el. A pow függvénynek kettő paramétere van, a hatványalap és a hatványkitevő, azaz pow(a,2). Ezen logika mentén eljutunk a végső megoldáshoz:

$$= (+(\text{pow}(a, 2), \text{pow}(b, 2), \text{pow}(c, 2))).$$

7.2.2. Prefix formula átalakítása prefix fává

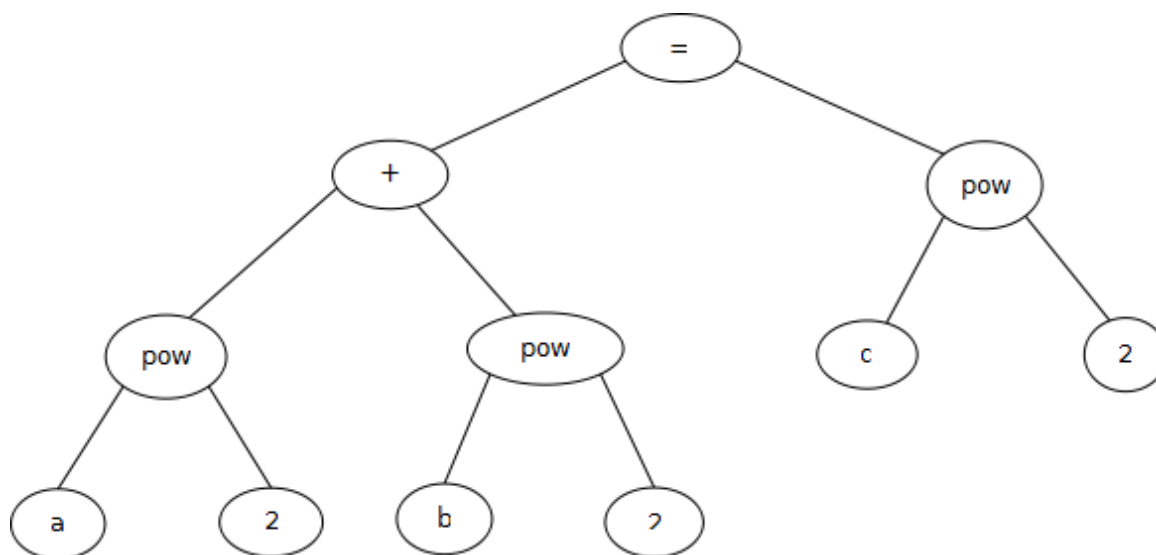
A formula átalakítása prefix fává meglehetősen egyszerű. A formula átalakítása bal- jobb irányban halad. A prefix formula első eleme a fában a gyökér elem lesz. A formulában lévő elem zárójelpárokkal rendelkezik. A zárójelpárokon belül megtaláljuk, hogy egy elem hány darab elemet tartalmaz. Ezek az elemek lesznek a gyerekelemek. A gyerekek a formulából kinyerően bal gyerek, jobb testvér bejárást biztosítanak a fán. A terminális elemek a fa legalsó szintjein helyezkednek el, nekik nincs gyerek elemük és levélnek nevezzük őket. Minden elemhez egy és csak is egy szülő rendelhető, kivéve a gyökér elemet, neki nincs szülője. A prefix fa vizuálisan is szemlélteti a dokumentáció struktúráját.

Példa a prefix nyelvből prefix fa alakításra:

Prefix forma:

$$= (+(\text{pow}(a, 2), \text{pow}(b, 2), \text{pow}(c, 2)))$$

Prefix fa:



9. ábra Példa a prefix formula prefix fává alakítására

Látható a példánkban, hogy balról az első elem, a gyökérelem az egyenlőség. Ezután a + elem következik, neki az = a szülője, őket összekötjük. Láthatjuk, hogy a + -nak van gyerek eleme. A bal oldali gyerekelem rögzítésére kerül sor. A következő vizsgálatunk arra tér ki, hogy neki van-e gyerek eleme. Az ő gyerek eleme a „a” és a „2” elemek, s a bal gyerek, jobb testvér bejárás elvét követve létrehozuk a következő szinten lévő csomópontokat. Terminális elemekhez érvén látjuk, hogy nekik nincs gyerekük, ezért az eggyel fentebbi szintre lépünk. Az új szinten megnézzük a formulából, hogy van-e testvére. A példánkban van, ezért elhelyezzük az új pow csomópontot, s rá is vizsgálatot végzünk az eddigiek alapján. Ezt a műveletet addig végezzük, amíg el nem fogy a formula és a teljes fát fel nem építjük.

7.2.3. Prefix fa átalakítása MathML

A prefix fa nem csak arra jó, hogy vizualitása miatt érthetőbbé tegye az átalakítás lépéseit az ember számára, hanem a feldolgozó program ennek segítségével könnyen elő tudja állítani az implementáláshoz szükséges MathML kódot. A feldolgozó megkapja a prefix fát, majd a csomópontokhoz az általunk megadott szabályok szerint MathML kódot társít. A művelet logikája: a feldolgozó megkapja a fát. A nulladik lépésben létrehoz egy `<mrow></mrow>` tag párt, ezután minden, a fában létező elem ezen két tag közé fog

kerülni. Ezután a gyökér elemet olvassa be, majd a megszokott bal gyerek, jobb testvér bejárást követi a fán. Minden csomópontozhoz érve megvizsgálja annak tulajdonságát és tartalmát. Ezekhez a csomópontokhoz a hozzá tartozó MathML tag-et rendelünk. Az operátorokhoz `<mo></mo>` tag-ek közé írja az operátor jelölő nevét (pl.: +,-). Amennyiben függvényt talál a fában akkor a szótárban lévő és a MathML kódban lévő társítások által megadja az `<mfüggvénynév></mfüggvénynév>` között annak tartalmát (pl. `mfrac`). Amikor terminális elemhez ér akkor két lehetőség előtt állunk. A levél elem vagy szám, vagy változó (betű). Szám esetében `<mn></mn>` tag, míg betű esetén `<mi></mi>` tag által bezárt tartalom jelenik meg.

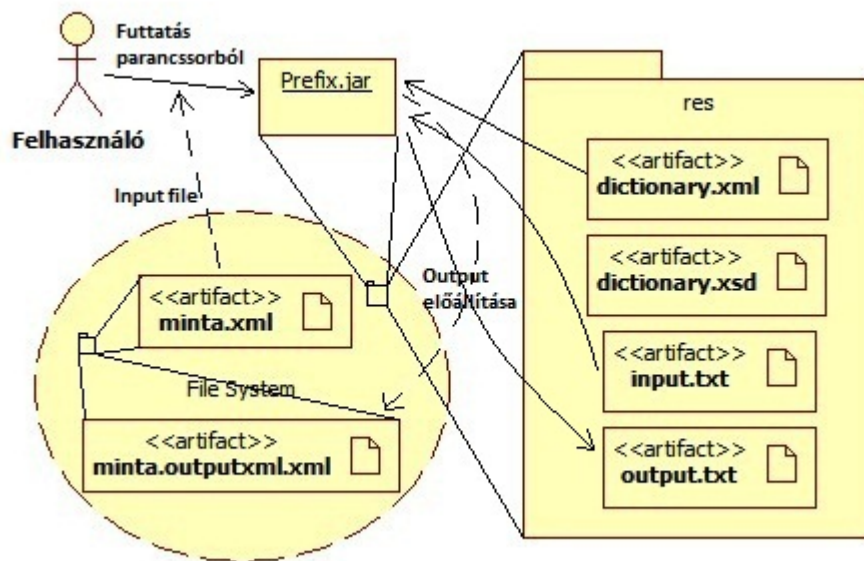
8. MINTARENDSZER IMPLEMENTÁCIÓJA

A kutatás során egy mintarendszer is implementálásra került, hogy megvizsgáljuk, mennyire ültethető jól át az elmélet a gyakorlatba.

8.1. Fejlesztői környezet, implementáció nyelve

Az implementáció során Java nyelvet használtunk a program kódjának megírásához. A programhoz tartozik egy külső szótár, ami XML formátumú (10. ábra). A szótár szabványos felépítéséhez (és a program megfelelő működésének biztosításához) egy sémát adtunk meg az XML dokumentum felépítésére, ezt XMLSchema szabvány szerint. A fejlesztés Eclipse fejlesztőkörnyezetben folyt.

Java nyelv használatát azért részesítettük előnyben, mert az egyik legelterjedtebb DocBook szerkesztőt, az Oxygen-t is ebben a nyelvben írták, és jövőbeni céljaink közt áll az is, hogy ehhez a szerkesztőhöz egy plugint készítsünk, ami már szerkesztés közben elkészíti a szükséges konverziókat.



10. ábra Az alkalmazás környezete

A *Prefix.jar* fájlt és a hozzá tartozó *res* mappát ugyanazon könyvtárban kell elhelyezni. A felhasználó parancssorból kétféleképpen tudja futtatni az alkalmazást. Az egyik lehetőség, hogy a felhasználó input paraméterként megad egy DocBook XML fájl relatív vagy teljes elérési útját és a nevét. Ebben az esetben a bemeneti fájlal azonos könyvtárban *fajlnev.outputxml.xml* néven jön létre a kimeneti állomány. A másik lehetőség, hogy a felhasználó nem ad meg paramétert, így az alkalmazás a *res* mappában található *input.txt* sorait veszi. Ezután az átalakított formulákat ugyanebben a mappában lévő *output.txt*-be írja. A program a konvertáláshoz szükséges szabályokat a *res* könyvtárban található *dictionary.xml* állományból olvassa be. Ennek a dokumentumnak a szerkezeti leírását a *dictionary.xsd* tartalmazza.

8.2. A szótár szerkezete

A szótár (*dictionary.xml*) központi szerepet tölt be az alkalmazásban. Ez az állomány tartalmazza a kifejezések formális leírását, a prefix nyelv elemeit és az átalakítás során létrehozott MathML kód vázát. A helyes megadásától függ az input értékek ellenőrzése, megfelelő átalakítása és a helyes output értékek előállítás.

Az alkalmazáshoz mellékelünk egy alap szótárat, amelyben már benne vannak a leggyakrabban használt szerkezetek. A csatolt fájlok között található egy *XMLSchema* fájl, amely egy szótár szerkezetét adja meg. Ennek segítségével egy XML szerkesztőben egyszerűen lehet a szótár dokumentumot felépíteni, megírni.

A szótár gyökéreleme az `<expressions>` elem. Közvetlenül ez alá kell felsorolnunk azokat az elemeket, melyek az alkalmazásnak majd ismernie kell.

Egy kifejezést `<Expression>` tag-ek közé kell megadni. Ennek az elemnek kötelező a `keyword` attribútumát kitölteni. Ez az attribútum írja le a prefix függvény nevét. Pontosan csak a zárójel előtti karaktereket adjuk meg, például: `plus(a,b)` a prefix függvény, a `keyword` attribútum értéke `plus`. Az `<Expression>` rendelkezik egy opcionális `operator` attribútummal is. Ennek értéke szigorúan csak egy karakter hosszú lehet, valamint a felvehető értékek halmaza is korlátozott, mindössze néhány aritmetikai, logikai operátor lehet. Ezt az attribútumot csak akkor használjuk, ha egy olyan prefix függvényt definiálunk, amelyet szeretnénk aritmetikai operátoroknál megszokott módon megadni a bemenetben. Például: `<Expression keyword="plus" operator="+">` hatására (persze szükséges a további mezők megadása is, de ezt később) az előfordító az $a+b$ kifejezést `plus(a,b)` alakra hozza, majd ezt a normalizált alakot adja át a transzformáló objektumoknak. A mintaalkalmazásban az előfordító átalakító funkciója korlátozott, mindössze a `+`, `-`, `*`, `/`, `=` operátorokat kezeli.

Az `<Expression>` tag első gyerekeleme a `<parameterSignature>` elem. Ez rendelkezik egy `numberOfParams` attribútummal, mely pozitív egész értékeket vehet fel, valamint az n karaktert. Megadása kötelező. Ez a szám a prefix függvény paramétereinek számosságát mutatja. Fix szám esetén pontosan ugyanannyi paraméterrel kell megadni a függvényt használatkor, n esetén tetszőleges paraméter számmal, de minimum 2 szükséges a biztosan helyes működéshez. A tetszőleges paraméterhossz az aritmetikai műveletek függvényei esetében hasznos, csökkenti a formula mélységét. E nélkül az $a+b+c$ formulát `plus(a,plus(b,c))` módon kellene leírni az egyszerűbb `plus(a,b,c)` helyett. A `<parameterSignature>` elemnek rendelkeznie kell szöveg tartalommal is. A szöveg a prefix függvény formális paraméter listája, zárójelekkel együtt. A formális paramétereket `@1`, `@2`, *stb.* jelöljük. Számozásukat mindig 1-től indítjuk, és eggyel növeljük. Szerepelhet benne `@n`, mely azt mutatja, hogy attól kezdve bármennyi paraméter előfordulhat (akár 0 is). Az utolsó `@n` paraméternek a feldolgozása azonosan zajlik.

Az `<Expression>` következő gyereke a `<MathML>` tag. Két gyereke lehet, `<PresentationMarkup>` (ennek megadása kötelező) és `<ContentMarkup>` (az alkalmazás ezt még nem kezeli). Ezekben szöveggént adhatjuk meg az előállítandó MathML kód paraméterezett változatát. Célszerű a MathML kódot `CDATA section`-ön belül megadni. A

MathML kódban szerepelnie kell minden paraméternek, amely a prefix függvény formális paraméter listájában szerepelt, de itt a sorrendet már nem kell megtartani, viszont figyeljünk a szimbólumokra: ami az alkalmazásban első paraméterként jött be (az @1 helyén) az a MathML kódban az @1 szimbólum helyére fog bekerülni, a második paraméter az @2 helyére, és így tovább. A 15. ábra mintáját használva mutatjuk be, hogyan jön létre az a kimeneti kód, melyben n a paraméterek megadható száma. A MathML formula megadásának a formája a példánk esetében: `<mrow>@1<mo>-</mo>@2<mo>-</mo>@n</mrow>`. A szótár beolvasásakor azokon a bejegyzéseken, ahol a paraméterek száma n végrehajtja a *PrefixDictionaryRecord* osztály *findIterativeSubstring* metódusát (megtalálja a megismételhető részt). Működésben annyit tesz, hogy a MathML kódban megkeresi az @ n szimbólumot, és a szimbólumot és az őt megelőző tag-et (XML szemléletben az őt megelőző testvért, bal testvért) egy sztringként eltárolja. Feldolgozás közben sorban halad a prefix függvény paraméterein. Amennyiben a paraméter sorszáma nagyobb, mint a MathML kódban található minta @ k minta, ahol k egy konkrét szám, akkor a paramétert @ n helyére illeszti be és utána bemásolja az *findIterativeSubstring* által lementett részt. Ha nincs több bemeneti paraméter, és a kimenet tartalmaz @ n szimbólumot, akkor ebből eltávolítja a megismételhető részt, és ez adja a kimenetet.

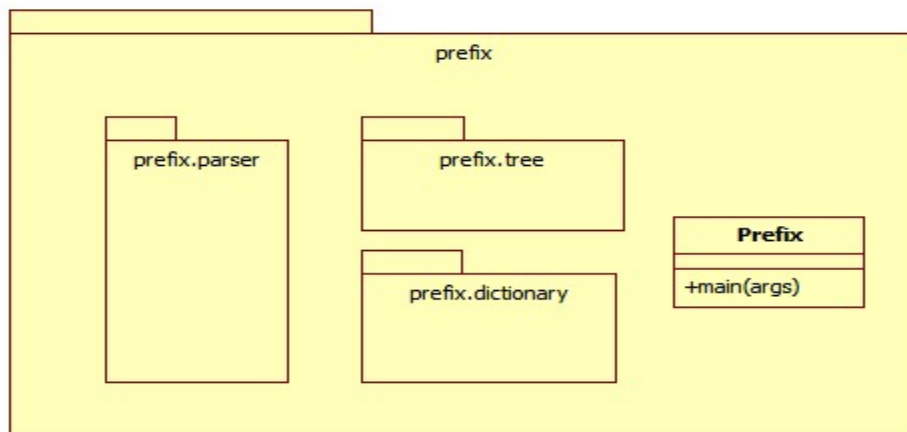
Figyeljünk arra, hogy a MathML kód megadása közben minden elemhez adjuk meg az *mml* prefixet, különben nem fog helyesen előállni a kimenet (előállítható így is, de ekkor még utólag bele kell nyúlni a DocBook dokumentum forrásába).

8.3. Az alkalmazás implementációs szerkezete

Az alkalmazás 3 jól elkülöníthető részből épül fel: a prefix fa helyes felépüléséért felelős részből, az átalakítási szabályokat beolvasó és megfelelő struktúrákká alakító részből, valamint a prefix fa tartalmát MathML nyelvre konvertáló részből.

A következő pontokban kicsit részletesebben bemutatjuk az egyes csomagok osztályainak feladatát, működési elvét. Teljes, közös osztálydiagramot a szűkös hely miatt nem mellékelünk.

A Prefix osztályban van az alkalmazás indulási pontja.

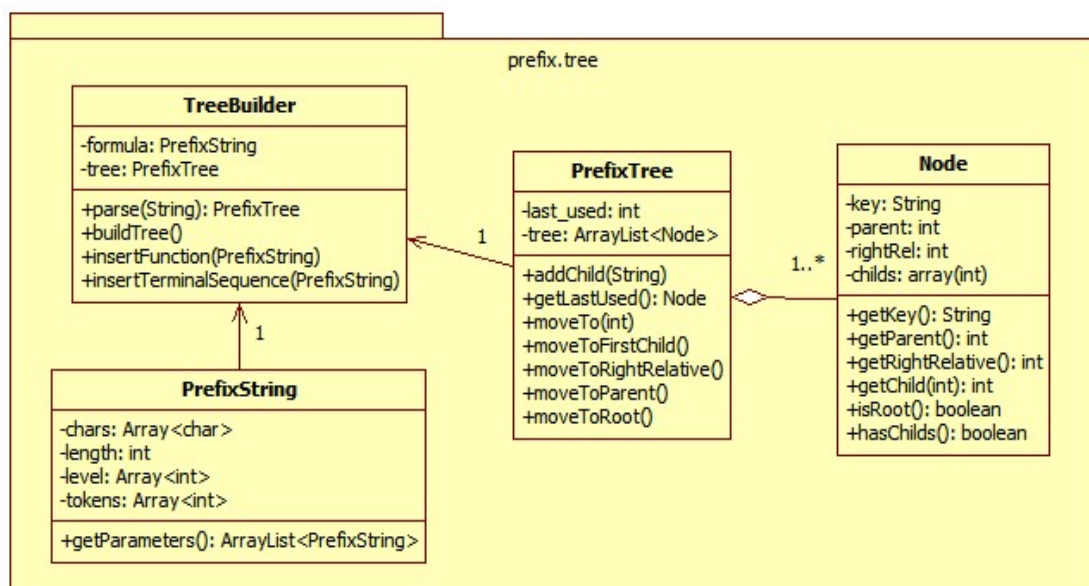


11. ábra Szerkezeti felépítés

8.4. prefix.tree

A *prefix.tree* csomag felelős azért, hogy egy bemeneti prefix formulából helyesen épüljön fel a prefix fa.

Az alábbi táblázatokban a *Node* osztály attribútumait és metódusait taglaljuk. Fő funkciója egy prefix függvény nevének vagy egy terminális elemnek az értékének eltárolása, valamint a szomszédos csomópontok nyilvántartása a navigáció érdekében. Gyakran hivatkozunk egy elem *indexére*. Ebben az esetben az indexet egy *PrefixTree* objektumnak *tree* listájában elhelyezett elemek indexén kell érteni.



12. ábra prefix.tree csomag osztálydiagramja

A Node osztály attribútumai

2. táblázat

Attribútum	Típus	Leírás, szerep
key	String	a csomópontban tárolt prefix függvény neve, vagy terminális elem értéke
parent	integer	a csomópont szülőjének indexe
rightRel	integer	a csomópont jobb oldali testvérének indexei
childs	integer[]	a csomópont közvetlen leszármazottainak indexei egy tömbben, a tömbben szereplő első index a baloldali gyereke a csomópontnak

A Node osztály metódusai

3. táblázat

Név	Argumentumok	Működés
Node	key: String	konstruktor, egy beadott kulcsértékkel előállít egy új Node-ot
getKey	-	visszaadja a csomópontban tárolt kulcs értékét
getParent	-	a csomópont szülőjének az indexét adja vissza
getRightRelative	-	a csomópont jobboldali szomszédjának indexét adja vissza
getChild	k: integer	a csomópont <i>k</i> -adik gyerekének az indexét adja vissza
isRoot	-	logikai értékkel tér vissza, igaz, amennyiben a csomópont gyökér a prefix fában, különben hamis
hasChilds	-	logikai értékkel tér vissza, igaz, amennyiben a csomópontnak van legalább egy gyerekeleme, különben hamis

A következő táblázatok a *PrefixTree* osztály attribútumait és metódusait írják le nagyvonalakban. Az osztály szerepe egy prefix fa tárolása, valamint a megfelelő bejárási rutinok biztosítása. A leírásban szerepel az *utoljára használt csomópont* kifejezés, ami arra a csomópontra utal, melynek indexét a *last_used* attribútumban tároljuk. A fában mozgást végző metódusok tulajdonképpen ezt az attribútumot módosítják.

A *PrefixTree* attribútumai

4. táblázat

Attribútum	Típus	Leírás, szerep
tree	ArrayList<Node>	a fában szereplő csomópontok listája
last_used	integer	a fa bejárásán, fán végzett művelet által utoljára érintett csomópont indexe

A *PrefixTree* metódusai

5. táblázat

Név	Argumentumok	Működés
addChild	key: String	a fához az utoljára használt (<i>last_used</i> által jelölt) csomópont alá beszúr egy új gyereket, gondoskodik a beszúrt csomópont és szomszédjai helyes indexbeállításairól
getLastUsed	-	visszaadja az utoljára használt csomópontot
moveTo	k: integer	a listában a <i>k</i> -adik indexű csomóponttra ugrik (ez lesz az utoljára érintett elem)
moveToFirstChild	-	az utoljára használt csomópont bal oldali gyerekeit állítja be utoljára használt elemként
moveToRightRelative	-	az utoljára használt csomópont jobb testvérét állítja be utoljára használt elemként
moveToParent	-	az utoljára használt csomópont szülőjét állítja be utoljára használt elemként
moveToRoot	-	a fa gyökerét állítja be utoljára használt elemként

A következő táblázatok a *PrefixString* osztály attribútumait és metódusait mutatják be. Az osztály egy prefix formulát tárol kiegészítő információkkal, amik jelentősen megkönnyítik a feldolgozást.

A *PrefixString* osztály attribútumai

6. táblázat

Attribútum	Típus	Leírás, szerep
chars	character[]	a tárolt prefix formula karaktereinek a tömbje
length	integer	a tárolt karakterlánc hossza
level	integer[]	karakterek szintjei, alapérték 0, de ha egy zárójelpár között lévő karakterek szintje zárójelpáronként egyel nő
tokens	integer[]	a karakterlánc speciális karaktereinek jelzésére készített tömb, olyan karakterek esetén van 0-tól különböző bejegyzés, mint: '(', ')', ',', ';'

A *PrefixString* osztály metódusai

7. táblázat

Név	Argumentumok	Működés
getParameters	-	egy formulában egy szinten lévő, elkülönülő rész-formulákat adja vissza amennyiben a formula egy terminális elemből áll, a 0-adik szinten lévő elemet adja vissza (maga a terminális) amennyibe a formula egy prefix függvény, akkor annak a függvénynek a vesszővel elválasztott elemeit adja vissza egy listában sorrendhelyesen összefűzve

Az alábbi táblázatok a *TreeBuilder* osztályról adnak rövid leírást. Az osztály feladata egy bemeneti szabályos prefix formulából egy prefix fa felépítése.

A *TreeBuilder* attribútumai

8. táblázat

Attribútum	Típus	Leírás, szerep
formula	PrefixString	szabályos prefix formulát tárol
tree	PrefixTree	a prefix formulából felépülő fát fogja tárolni

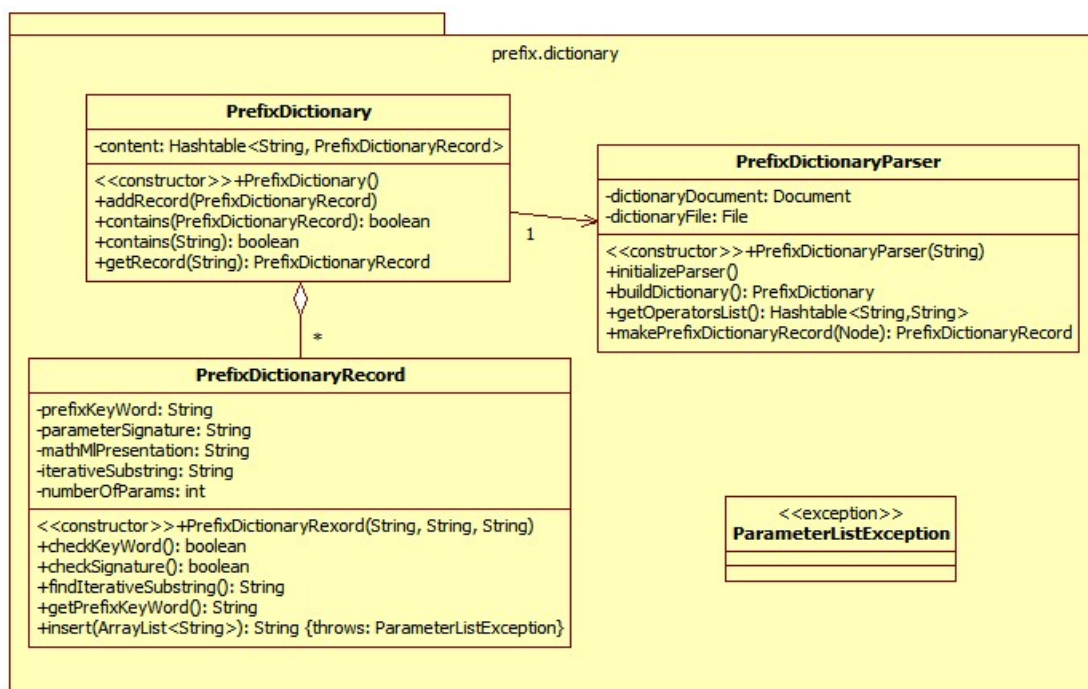
A *TreeBuilder* metódusai

9. táblázat

Név	Argumentumok	Működés
parse	formula: String	leellenőrzi, hogy a bemeneti sztring szabályos prefix formula-e, majd eltárolja az objektum <i>formula</i> attribútumában és meghívja a <i>buildTree</i> metódust
buildTree	-	az objektumban tárolt <i>formula</i> -ból felépít egy prefix fát, amit az objektum <i>tree</i> attribútumában tárol le működése annyi, hogy a bemeneti formulára meghívja az <i>insertFunction</i> metódust
insertFunction	formula: PrefixString	rekurzív metódus, amennyiben a bemenete terminális, meghívja az <i>insertTerminalSequence</i> metódust, különben beszúrja magát (<i>PrefixTree addNode</i> metódusával) és saját magát hívja meg a formula külső prefix függvényének összes paraméterére
insertTerminalSequence	terminal: PrefixString	egy levelet illeszt be a prefix fába, amely értéke a bemeneti terminális elem

8.5. prefix.dictionanry

A prefix.dictionary csomag osztályainak feladata, hogy az XML fájlban tárolt szótárból beolvassa a bejegyzéseket, és azt más osztályok által használható objektumokká alakítsa.



13. ábra prefix.dictionary csomag osztálydiagramja

A következő táblázatok a *PrefixDictionaryRecord* osztály attribútumait és metódusait mutatja be.

A *PrefixDictionaryRecord* osztály attribútumai

10. táblázat

Attribútum	Típus	Leírás, szerep
prefixKeyWord	String	egy prefix függvény nevét tárolja
parameterSignature	String	egy prefix függvény paraméterlistáját tárolja (a zárójelpárok közti sztring, zárójelekkel együtt)
mathMIPresentation	String	egy MathML képlet vázát tárolja, főleg ebből állítja elő a kimenetet
iterativeSubstring	String	definiálhatunk olyan prefix függvényeket, melyek n darab paraméterrel rendelkeznek helyes szótár megadás esetén egy bejegyzés objektum ebbe az attribútumba szedi ki az ebben az esetben megismétlendő MathML kódot
numberOfParams	integer	egy prefix függvényhez tartozó paraméterek száma n darab paraméter esetén 999 az értéke

A *PrefixDictionaryRecord* osztály metódusai

11. táblázat

Név	Argumentumok	Működés
PrefixDictionaryRecord	konstruktor, String, String, String	inicializál egy PrefixDictionaryRecord objektumot a paramétereit sorban: prefix függvény neve, prefix függvény paraméter szignatúrája, a prefix függvényhez tartozó MathML kód váza (primitív)
checkKeyWord	-	ellenőrzi, hogy a prefix függvény neve megfelel-e az előírásoknak amennyiben megfelel, logikai igaz, különben hamis értéket ad vissza
checkSignature	-	ellenőrzi, hogy a prefix függvény (primitív) paraméter szignatúrája megfelel-e az előírásoknak amennyiben megfelel, logikai igaz, különben hamis értéket ad vissza
findIterativeSubstring	String	a MathML kód primitívjében megkeresi a megismétlendő részt a metódus csak akkor fut le, ha a <i>numberOfParams</i> = 999 visszatérési értéke a megtalált string
getPrefixKeyWord	String	visszatérési értéke az objektumban tárolt prefix függvény neve

Név	Argumentumok	Működés
insert	ArrayList<String>	jelentős szerepe van a kimenet előállításában bemeneti egy string lista, melyek a prefix függvény paraméterire illesztett sztringek a metódus ezeket a sztringeket illeszti be a MathML kód megfelelő helyeire amennyiben a bemeneti paraméterek száma (a lista hossza) kevesebb, mint a minimum (megismételhető résszel rendelkező prefix függvény esetén), vagy eltér a megadottól (fix számú paraméterrel rendelkező prefix függvény) akkor a metódus <i>ParameterListException</i>-t dob

A következő táblázatok a *PrefixDictionary* osztály tulajdonságait, viselkedését részletezik. Az osztály feladata több *PrefixDictionaryRecord* objektum tárolása és ezek a hatékony keresés biztosítása.

A *PrefixDictionary* osztály attribútumai

12. táblázat

Attribútum	Típus	Leírás, szerep
content	Hashtable<String, PrefixDictionaryRecord>	a szótár tartalma a Java nyelv által biztosított <i>Hashtable</i> osztály megfelelő keresési időt biztosít a feladatunkhoz a kulcs, ami azonosítja a struktúrában lévő objektumokat a prefix függvény neve, az objektumok pedig szótári bejegyzések

A *PrefixDictionary* osztály metódusai

13. táblázat

Név	Argumentumok	Működés
PrefixDictionary	konstruktor	inicializál egy PrefixDictionary objektumot, előkészíti a <i>content</i> attribútumot a használatra
addRecord	PrefixDictionaryRecord	egy új bejegyzést tesz a szótárba elkerüli, hogy egy kulcshoz több bejegyzés is tartozzon
contains	PrefixDictionaryRecord	logikai igaz értékkel tér vissza, ha a paraméterben szereplő, vele megegyező attribútumokkal rendelkező bejegyzés már van a szótárban, különben hamis
contains	String	igaz logikai értékkel tér vissza, ha a bemeneti string megtalálható a <i>content</i> kulcsai között, különben hamis
getRecord	String	bemeneti paraméterben megadott kulcsú szótárbejegyzést keresi ki visszatérési értéke ez az objektum

Az alábbi táblázatok a *PrefixDictionaryParser* osztály attribútumait és metódusait mutatja be. Az elsődleges feladata egy XML fájlból előállítani osztály egy *PrefixDictionary* típusú objektumot további használatra.

A *PrefixDictionaryParser* osztály attribútumai

14. táblázat

Attribútum	Típus	Leírás, szerep
dictionaryDocument	org.w3c.dom.Document	DOM szabványnak megfelelő Document objektum a beolvasott XML fájl tartalmát tárolja
dictionaryFile	File	egy fájl pointer, ami a számítógépen lévő szótár XML fájlra mutat

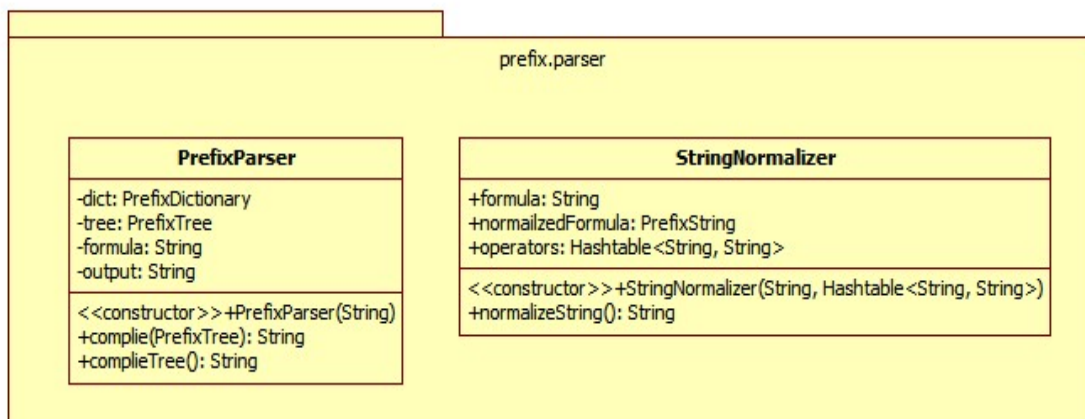
A *PrefixDictionaryParser* osztály attribútumai

15. táblázat

Név	Argumentumok	Működés
PrefixDictionaryParser	konstruktor, String	inicializál egy PrefixDictionaryParser objektumot a paraméterében megkapott string a szótár XML fájl URL-je
initializeParser	-	előkészíti az objektumot a szótár előállításához
buildDictionary	-	a megadott szótár fájlból elkészít egy PrefixDictionary objektumot további használatra itt fűzi össze a bejegyzéseket szótárrá visszatérési értéke az elkészült szótár
getOperatorList	-	operátorként használható prefix függvények nevét és operátorát adja vissza egy Hashtable objektumban (StringNormalizer számára)
makePrefixDictionaryRecord	w3c.org.dom.Node	egy dokumentum csomópontból előállít egy <i>PrefixDictionaryRecord</i> objektumot, amit hozzá tud fűzni a <i>PrefixDictionary</i> -hez

A *ParameterListException* osztály egy speciális kivétel osztály. A *PrefixDictionaryRecord* ilyen típusú kivételt generál, ha nem megfelelő hosszúságú az a paraméter lista, amit a bejegyzésében tárolt prefix függvény primitív megkövetelne.

8.6. prefix.parser



14. ábra prefix.parser csomag osztálydiagramja

A *prefix.parser* csomag összefogja az eddig felsorolt osztályokat, ezáltal egy kényelmes, kevés utasítást tartalmazó interfészt létrehozva, mely a megfelelő működést biztosítja a lehető legkevesebb kód segítségével. Ez azt is magában hordozza, hogy az működés részletei rejtve maradnak előttünk.

A következő táblázatok a *PrefixParser* osztály metódusait és attribútumait mutatja be. A bemeneti string tárolására azért van szükség, mert a prefix fává alakítás előtt szabályosság vizsgálatára és előzetes átalakításokra lehet szükség. A kimeneti string eltárolása lehetővé teszi az alkalmazás későbbi verzióiban különböző típusú, szerkezetű kimenetek generálására.

A *PrefixParser* osztály attribútumai

16. táblázat

Attribútum	Típus	Leírás, szerep
dict	PrefixDictionary	az alkalmazás futása közben használt szótárat tárolja
tree	PrefixTree	az a bemeneti prefix formulából létrehozott prefix fa
formula	String	az bemeneti formula tárolása
output	String	a kimeneti MathML kód tárolására

A *PrefixParser* osztály metódusai

17. táblázat

Név	Argumentumok	Működés
PrefixParser	konstruktor, String	inicializál egy PrefixParser objektumot a bemeneti sztringet az objektum formulájába teszi, elkészíti a prefix fát
comple	PrefixTree	rekurzív metódus, az inputjában megkapott rekurzív metódust egyre kisebb részfákra bontja a csomópontokon előállítja a megfelelő MathML kimenetet
compleTree	-	az objektumban tárolt prefix fából előállítja a MathML kimenetet

8.7. Használati útmutató

A dolgozat mellékletében található CD-n megtalálható a mintaalkalmazás *Prefix.jar* néven. Futtatásához *Java Runtime Enviroment* szükséges, valamint, hogy az állomány mellett legyen a CD-n is megtalálható *res* mappa. Ez a mappa tartalmazza a szótárként funkcionáló XML dokumentumot (*dictionary.xml*). A *res* mappa tartalmaz továbbá egy XSD kiterjesztésű XMLSchema fájlt (*dictionary.xsd*), amely a szótár szerkezetének leírását tárolja.

```

<Expression keyword="minus" operator="-">
  <ParameterSignature numberOfParams="n">(@1,@2,@n)</ParameterSignature>
  <MathML>
    <PresentationMarkup><![CDATA[
      <mrow>
        @1
        <mo>-</mo>
        @2
        <mo>-</mo>
        @n
      </mrow>]]</PresentationMarkup>
    </MathML>
  </Expression>

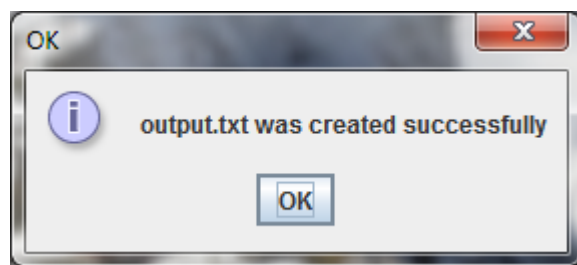
```

15. ábra Egy szótár bejegyzés mintája

A 15. ábra mutatja, hogyan néz ki egy szótár bejegyzés az XML fájlban. A *keyword* attribútum tartalmazza a prefix függvény nevét, az *operator* paraméter opcionális, és csak korlátozott értékeket vehet fel. Mind a két attribútumnak egyedinek kell lennie. Az

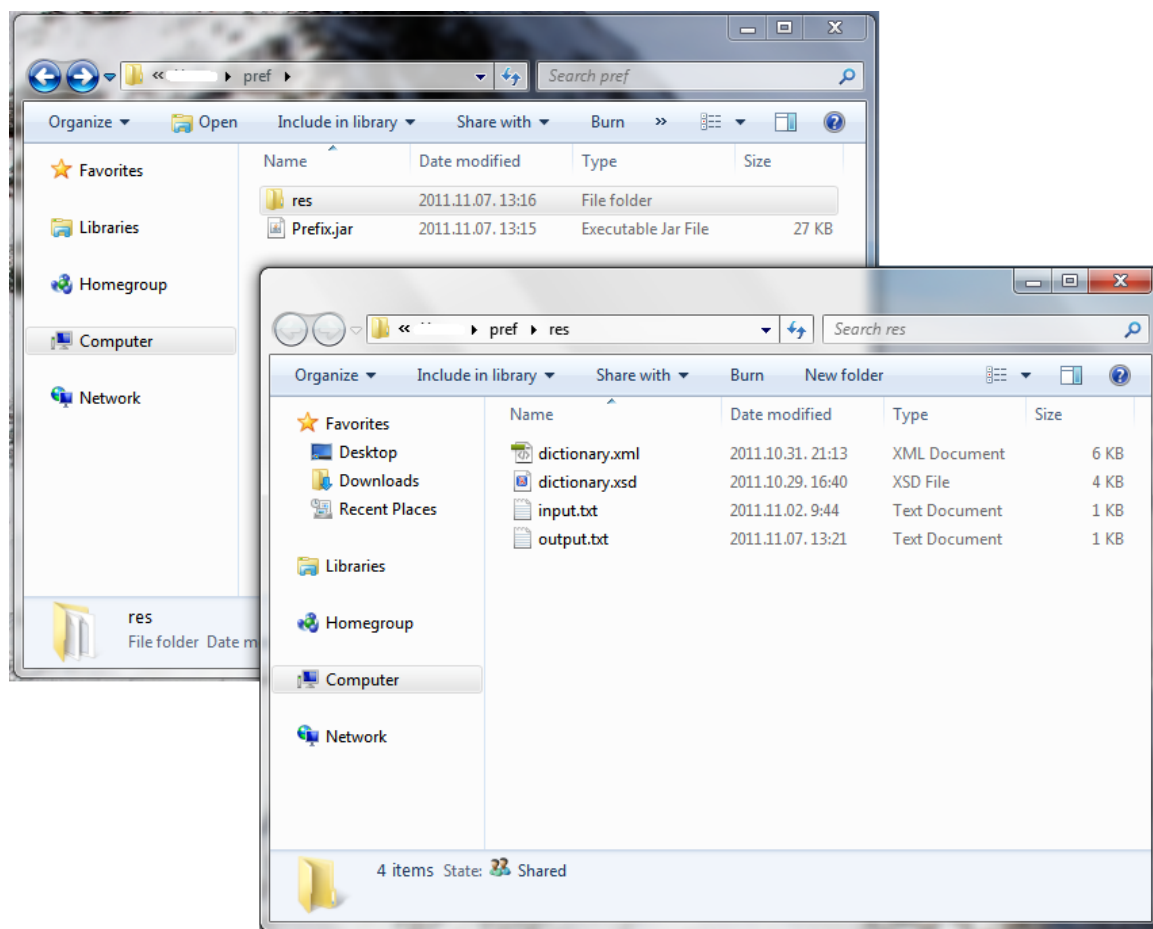
operator attribútum arra szolgál, hogy az alkalmazás előfordítója a természetes módon megadott aritmetikai műveleteket a megfelelő módon prefix függvénné alakítsa. Ehhez az átalakításhoz is a szótár bejegyzéseit használja. A *parameterSignature* elem szövegtartalma a prefix függvény paraméterlistájának zárójelekkel ellátott vagy pontos mása, vagy a minimum paraméterek és egy *@n* szimbólum. A paramétereket *@1*, *@2*, *@3* stb. szimbólumokkal jelöljük, 1-gyel kezdődően, növekvő sorrendben. Az *@n* szimbólum jelképezi, hogy a paraméterek száma tetszőleges lehet. Az elem *numberOfParams* attribútuma szám szerint is megadja a paraméterek lehetséges számát, lehetséges értékei pozitív egészek és az *n*. A *MathML* elem kétféle megengedett gyerekkel rendelkezhet: *PresentationMarkup* (kötelező a megadása) valamint *ContentMarkup*. Az utóbbit az alkalmazás még nem használja. A *PresentationMarkup* elembe írhatjuk le azt a *MathML* kódot, amit az alkalmazásunknak elő kell állítania. Fontos, hogy a *MathML* kódot mindig *<mrow>* *</mrow>* tagek közé írjuk, elkerülve ezzel az esetleges megjelenítési problémákat. Azokra a helyekre, ahová a prefix függvény paraméterei lesznek beillesztve, oda a megfelelő *@k* szimbólumot kell írni, ahol *k < numberOfParams* (tehát szerepelt a *parameterSignature*-ban). A *MathML* kódon belül már tetszőlegesen helyezhetjük el a szimbólumokat, az alkalmazás gondoskodik arról, hogy minden a megfelelő helyre kerüljön (tehát ami első paraméterként jön be, az az *@1* helyre kerül). Érdekes a *MathML* kódot *CDATA Section*-ba írni, így megkönnyítve saját dolgunkat. A szótár bővítése után érdemes a fájlt az XSD segítségével *validálni*.

Az alkalmazást a következő képen kell használni: a JAR fájl melletti *res* mappában találunk egy *input.txt* szöveges fájlt (17. ábra). Ennek a soraiban kell megadnunk a prefix formulákat (soronként egyet), majd futtatjuk a *Prefix.jar*-t.



16. ábra Sikeres futtatás 1

A futtatás után a *res* mappa *output.txt* fájlban lesznek a generált kódok. Futás sikerességét a program egy felugró ablakkal jelzi (16. ábra).



17. ábra A mellékelt fájlok elhelyezkedése

Lehetőségünk van parancssorból is futtatni az alkalmazást, a parancsokat a 18. ábra mutatja.

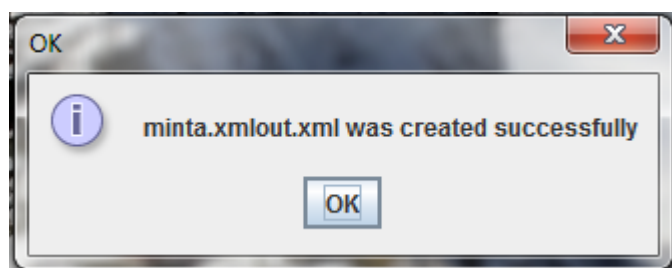
```

Administrator: C:\Windows\system32\cmd.exe
C:\Users\pref>java -jar Prefix.jar
C:\Users\pref>java -jar Prefix.jar "C:\minta.xml"
C:\Users\pref>_

```

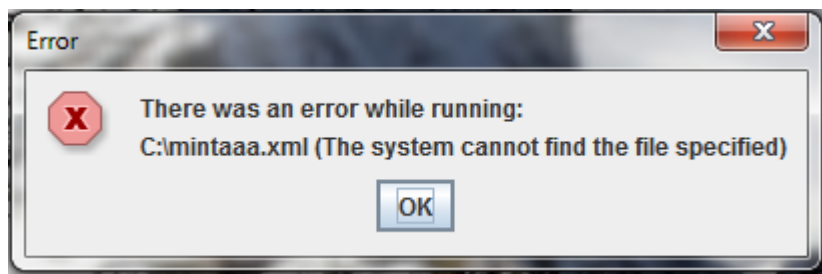
18. ábra Futtatás parancssorból

A `java -jar Prefix.jar` parancs kiadásával ugyanaz a futási eredmény a fentebb leírtakkal. A `java -jar Prefix.jar „URL”` parancsban az *URL* helyére egy XML kiterjesztésű DocBook fájl teljes, vagy relatív (a *Prefix.jar*-hoz képest) elérési útvonalát és a fájl nevét (kiterjesztéssel együtt) adhatjuk meg. A dokumentumban a MathML kódok helyére (`<math></math>` tagek közé) kell beírni a prefix formulát. Az alkalmazás futás közben a formulákat kicseréli a megfelelő MathML részekre, az így elkészült dokumentumot az *URL*-ben megadott mellé menti. Ha a bemeneti fájl neve *minta.xml* akkor a keletkező dokumentumé *minta.xmlout.xml*. A futás sikerességéről értesítést kapunk (19. ábra).



19. ábra Sikeres futtatás 2

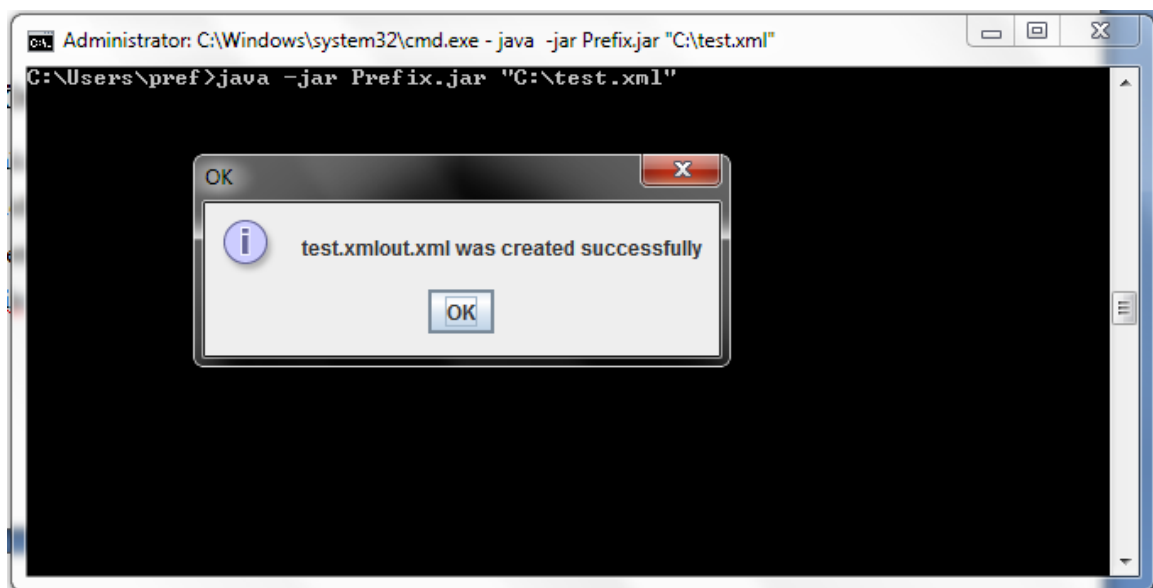
Amennyiben az alkalmazás működése fellépő problémákról egy hibaüzenetben kapunk tájékoztatást (20. ábra).



20. ábra Hibaüzenet

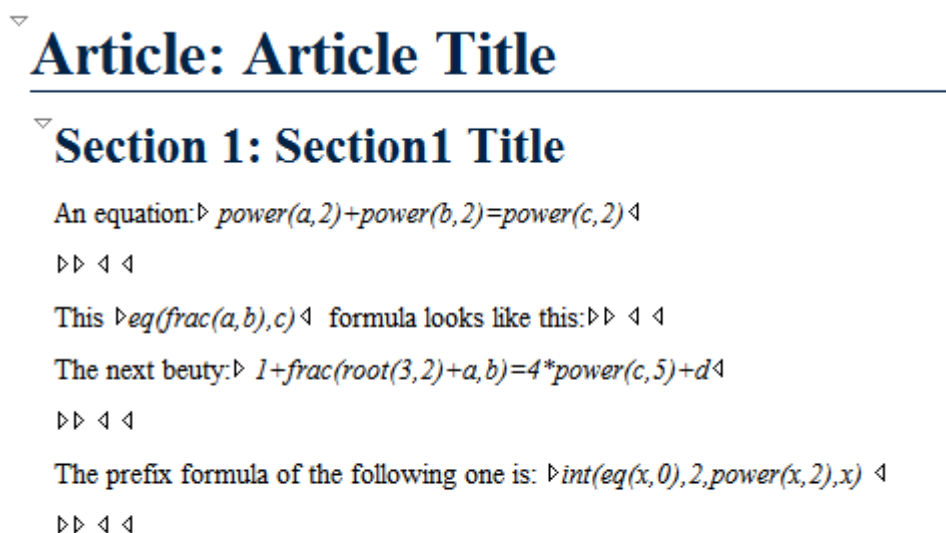
9. TESZTELÉS

Az alábbiakban ábrák segítségével mutatjuk be a létrehozott nyelv működését lépésről lépésre. Az 21. ábra látható a parancssori parancs, amelynek segítségével futtathatjuk a fájlunkat. A felugró ablak jelzi, hogy a futtatás sikeres volt. A *test.xml* is megtalálható az alkalmazás melletti *res* könyvtárban, valamint a dokumentum mellékleteik közt is (2. sz. melléklet).



21. ábra A teszt futási eredménye

A 22. ábra megmutatja a *test.xml* fájl képét. Ezt a formát az Oxygen XML Editor DocBook szerkesztőjének *Author* megjelenítése, melyet ez az alkalmazás szerkesztés közben mutat. Ez a felület tulajdonképpen egy WYSIWYM (What You See Is What is in Your Mind) felület. A 23. ábra ennek a fájlnek a forrását mutatja.



22. ábra A bemeneti DocBook dokumentum képe

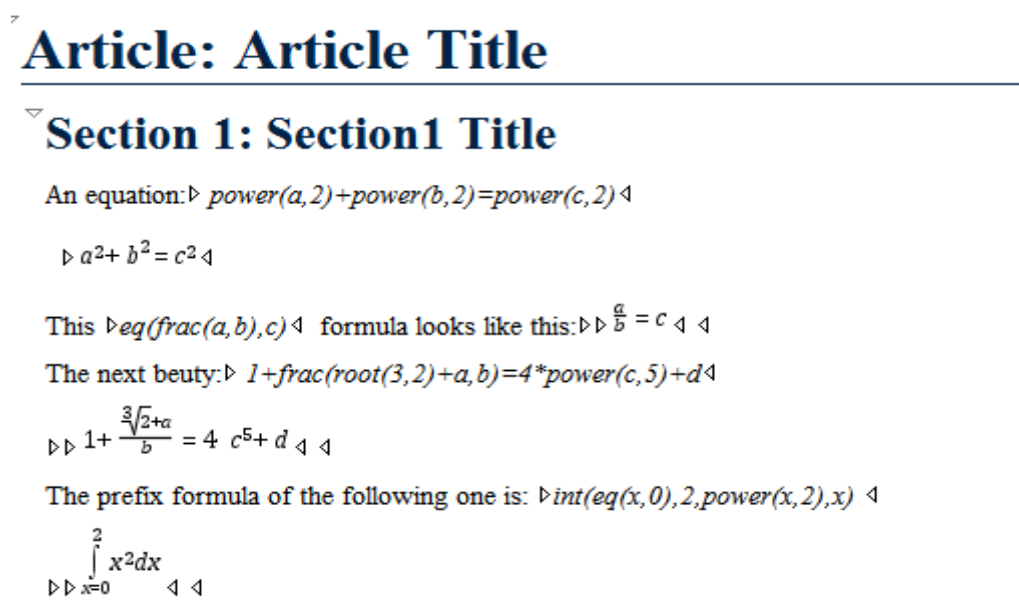
```

3 <article xmlns="http://docbook.org/ns/docbook">
4   <title>Article Title</title>
5   <sect1>
6     <title>Section1 Title</title>
7     <para>An equation:<emphasis role="italic">
8       power(a,2)+power(b,2)=power(c,2)</emphasis></para>
9     <para><inlineequation>
10       <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
11         power(a,2)+power(b,2)=power(c,2)</mml:math>
12       </inlineequation></para>
13     <para>This <emphasis role="italic">eq(frac(a,b),c)</emphasis> formula looks like this:<inlineequation>
14       <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"> eq(frac(a,b),c)
15       </mml:math>
16     </inlineequation></para>
17     <para>The next beauty:<emphasis role="italic">
18       1+frac(root(3,2)+a,b)=4*power(c,5)+d</emphasis></para>
19     <para><inlineequation>
20       <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
21         1+frac(root(3,2)+a,b)=4*power(c,5)+d</mml:math>
22       </inlineequation></para>
23     <para>The prefix formula of the following one is: <emphasis role="italic">
24       >int(eq(x,0),2,power(x,2),x)</emphasis></para>
25     <para><inlineequation>
26       <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
27         int(eq(x,0),2,power(x,2),x)</mml:math>
28       </inlineequation></para>
29   </sect1>
30 </article>

```

23. ábra A bemeneti DocBook dokumentum forrása

A 24. ábra mutatja a *Prefix.jar* által előállított *tes.outputxml.xml* fájl képe szintén az Oxygen szerkesztőt *Author* felületén. Ezen látható, hogy a látvány megfelel várakozásainknak. A 25. ábra egy részletet mutat ennek az állománynak a kódjából. Az alkalmazásunk teljes kód szintén megtalálható a mellékletek között (3. sz. melléklet).



24. ábra Az elkészített dokumentum képe

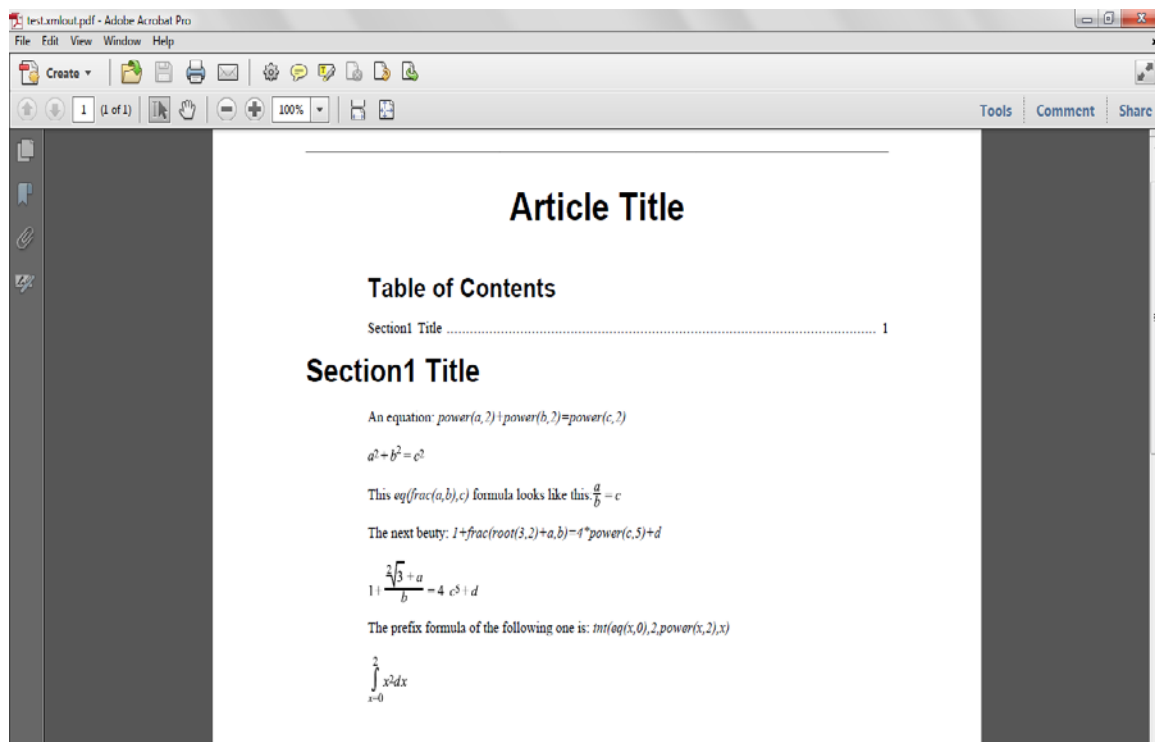
```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <?oxygen RNGSchema="http://www.oxygenxml.com/docbook/xml/5.0/rng/dbmathml.rng" type="xml"?>
3
4  <article xmlns="http://docbook.org/ns/docbook">
5    <title>Article Title</title>
6    <sect1>
7      <title>Section1 Title</title>
8      <para>An equation:<emphasis role="italic"> power(a,2)+power(b,2)=power(c,2) </emphasis><equation>
9        <math xmlns="http://www.w3.org/1998/Math/MathML">
10          <mrow>
11            <mrow>
12              <msup>
13                <mrow>
14                  <mi>a</mi>
15                </mrow>
16              </msup>
17              <mn>2</mn>
18            </mrow>
19            <mo>+</mo>
20            <msup>
21              <mrow>
22                <mi>b</mi>
23              </mrow>
24              <mn>2</mn>
25            </msup>
26            <mn>=</mn>
27            <msup>
28              <mrow>
29                <mi>c</mi>

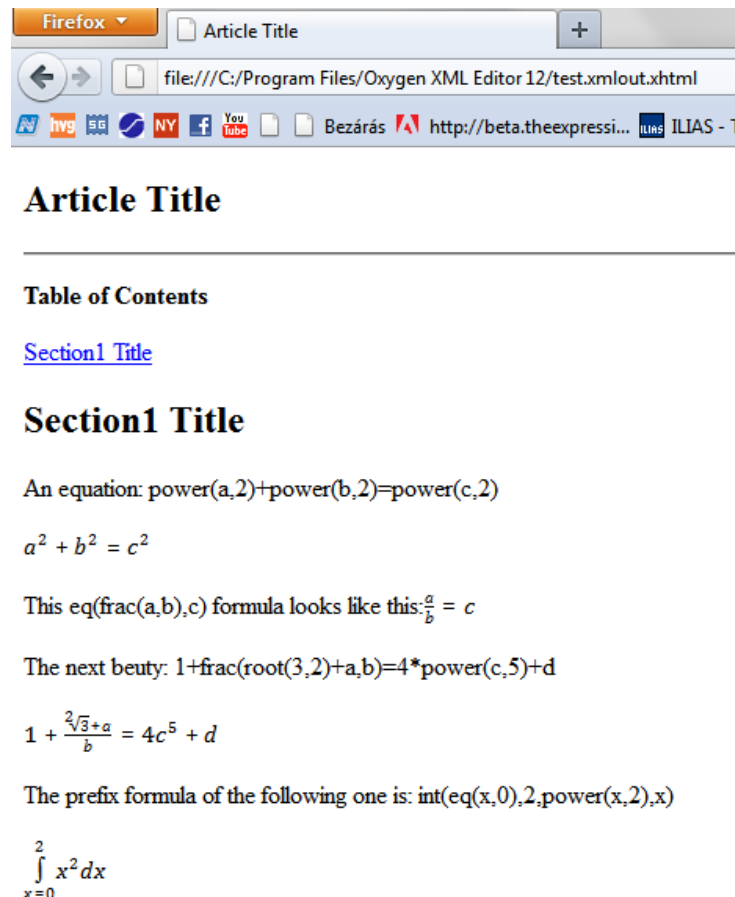
```

25. ábra A kimeneti dokumentum forrása (részlet)

A program előállít felesleges csomópontokat is, egy gyerekelemmel rendelkező `<mrow>`-kat. Ez nem okoz problémát, de felesleges adat, ezért az alkalmazás későbbi verzióiban javítva lesz.



26. ábra A `test.outputxml.xml` PDF-re konvertált alakja



27. ábra A *test.outputxml.xml* XHTML-re konvertált alakja (Mozilla Firefox 7 böngészőben)

A 26. ábra és 27. ábra mutatja a keletkezett DocBook XML állomány PDF illetve XHTML formátumokra konvertált alakja. A képletek kinézete megfelelt előzetes elvárásainknak. A formulák formája, alakja a DocBook szabvány konverzióján múlik, ami alkalmazásunkkal mindössze a képlet megadás módja egyszerűsödik. A *test.outputxml.xml* állomány PDF-re és XHTML-re konvertált változata egyaránt megtalálható a *res* mappában.

10. ÖSSZEGZÉS ÉS KITTEKINTÉS

Munkánk során sikerült elérni a kitűzött célok megoldását, megvalósítását. A javasolt modell könnyebbé, egyszerűbbé teszi a felhasználó számára, hogy a hosszas kódolást felváltsa egy gyors egyszerű munkamenet. A kidolgozott rendszer magában foglalja egy új, matematikai képletből való prefix nyelv létrehozását. Erre építve létrehoztuk a prefix fát, amit egyszerűen előállíthatunk a prefix nyelvből. A logikai fa segítségével és a szótár

használatával az alkalmazás elkészíti a megfelelő MathML kimenetet. A szótár tartalmát a mellékelt XMLSchema segítségével töltöttük fel, amelyet a felhasználó bizonyos keretek között bővíthet. Az eljárásunk egy Java nyelven készített grafikus kezelői felületet kapott, ezzel biztosítva a felhasználók számára a könnyű kezelhetőséget, használatot. Kutatásunk nem tekinthető véglegesen lezárult projektnek. A későbbiekben szeretnénk bővíteni, szélesíteni a projektünket. Dolgozatunkban végig törekedtünk az egyszerű, logikus, könnyen érthető leírásra, az átláthatóságra. Az eddigi eredményeink remélhetőleg minél szélesebb körben lesznek használatosak. Az elvégzett tevékenységet nemcsak az elért eredmények miatt tartjuk hasznosnak, hiszen a projekt által egy részletes, mély betekintést nyerhettünk az informatikai világ különböző szegmenseibe, amivel tanulmányaink alatt nem találkoztunk volna. Ilyen terület például a formális nyelvek és automaták területe, a különböző szövegszerkesztők megismerése, használata. Külön öröm számunkra, hogy tanulmányaink alatt részt vehettünk egy szakirányú projektben, tapasztalatokat szerezhettünk egy szoftver fejlesztésének menetében, amihez saját ötleteinket felhasználva kaptuk meg a végterméket.

10.1. Lehetőségek

A prefix nyelv általánossága miatt alkalmas lehet más matematikai képletek leírására szolgáló szabvány megadására. Megoldható segítségével egy leírásból LaTeX képlet létrehozása is, mindössze a konverziós szabályokat kell finomra hangolni.

Továbbgondolva az alkalmazást, a prefix nyelv alkalmas lehet a LaTeX és a MathML formulák közti oda-vissza konvertálásra. A nyelv szerepe itt egyfajta köztes állapotot írna le, melyből mindkét kimenet könnyen generálható.

10.2. Oxygen plugin

Az Oxygen XML szerkesztő e legelterjedtebb szerkesztő DocBook formátum esetén. Ezt az alkalmazást Java nyelvben fejlesztik, bővíthetősége, pluginok készítése egyszerű. A fejlesztők jól dokumentálták az alkalmazás API-ját, ezáltal lehetővé téve az egyszerű, mindenki számára elérhető bővíthetőséget.

Mivel a mintaalkalmazásunk is Java nyelvben készült, elég csak kibővíteni a megfelelő csomagokkal és a jó használhatóság érdekében egy grafikus felülettel, hogy egy megfelelő, mindenki számára használható plugint kapjunk.

Ez jelentősen megkönnyíteni a dokumentumok készítését: nem kell egyszerre az egész dokumentumot lefordítani, így könnyebb a hibák javítása. A képletek kinézetét már a beírás után láthatjuk, nincs szükség külön parancsokra.

TÁMOGATÓK:

A bemutatott TDK dolgozat a TÁMOP-4.2.1.B-10/2/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

11. IRODALOMJEGYZÉK

- [1] Bach I. [2002] Formális nyelvek. Budapest, 13-26. o.
- [2] Bujdosó Gy., Fazekas A. [1996]: TeX kezdőlépések, Budapest, 55-70. o.
- [3] Csárdi G.: [1998] LaTeX2e 69 percben. In:
http://csomalin.csoma.elte.hu/~flu/konyvek/lkh_22.html (letöltve:
2011.11.01.)
- [4] Knuth, D. E. [1984]: The TeXbook, Reading, Massachusetts, 7-11.o., 127-137. o.
- [5] Walsh, N. [2010]: DocBook 5: The Definitive Guide. In:
<http://www.docbook.org/tdg51/en/html/> (letöltve: 2011.11.02.)
- [6] Partl, H. – Hyna, – I., Schlegl, E [2011] The Not So Short Introduction to LaTeX2e. In: <http://ftp.roedu.net/mirrors/ctan.org/info/lshort/english/lshort-a5.pdf> (letöltve: 2011.11.01.) 51-81. o.
- [7] Wettl F. – Mayer Gy. – Szabó P. [2004]: LaTeX kézikönyv, Budapest, 1-2. o., 397-412. o.
- [8] W3C Recommendation: <http://www.w3.org/TR/MathML2/> (letöltve: 2011.11.02.)
- [9] <http://docbook.org/whatis> (letöltve: 2010.11.02.)
- [10] <http://en.wikipedia.org/wiki/DocBook> (letöltve: 2011.11.01.)
- [11] <http://en.wikipedia.org/wiki/TeX> (letöltve: 2011.10.30)
- [12] <http://en.wikipedia.org/wiki/XMLSpy> (letöltve: 2011.11.02.)
- [13] <http://hu.wikipedia.org/wiki/LaTeX> (letöltve: 2011.11.01.)
- [14] http://hu.wikipedia.org/wiki/Microsoft_Word (letöltve: 2011.11.03.)
- [15] <http://hu.wikipedia.org/wiki/WYSIWYG> (letöltve: 2011.11.03.)

MELLÉKLETEK

1. számú melléklet:

A 17. oldal található prefix formula átalakítási lépései.

$$\begin{aligned} S \rightarrow L \rightarrow F \rightarrow P(A \rightarrow & eq(A \rightarrow eq(S, A \rightarrow eq(S, A \rightarrow eq(L, A \rightarrow eq(F, A \rightarrow \\ & \rightarrow eq(P(A, A \rightarrow eq(plus(A, A \rightarrow eq(plus(S, A, A \rightarrow eq(plus(L, A, A \rightarrow \\ & \rightarrow eq(plus(F, A, A \rightarrow eq(plus(P(A, A, A \rightarrow eq(plus(pow(A, A, A \rightarrow \\ & \rightarrow eq(plus(pow(S, A, A, A \rightarrow eq(plus(pow(L, A, A, A \rightarrow \\ & \rightarrow eq(plus(pow(V, A, A, A \rightarrow eq(plus(pow(a, A, A, A \rightarrow \\ & \rightarrow eq(plus(pow(a, S), A, A \rightarrow eq(plus(pow(a, N), A, A \rightarrow \\ & \rightarrow eq(plus(pow(a, 2), A, A \rightarrow eq(plus(pow(a, 2), S), A \rightarrow \\ & \rightarrow eq(plus(pow(a, 2), L), A \rightarrow eq(plus(pow(a, 2), F), A \rightarrow \\ & \rightarrow eq(plus(pow(a, 2), P(A), A \rightarrow eq(plus(pow(a, 2), pow(A), A \rightarrow \\ & \rightarrow eq(plus(pow(a, 2), pow(S, A), A \rightarrow eq(plus(pow(a, 2), pow(L, A), A \rightarrow \\ & \rightarrow eq(plus(pow(a, 2), pow(V, A), A \\ & \rightarrow eq(plus(pow(a, 2), pow(b, A), A \\ & \rightarrow eq(plus(pow(a, 2), pow(b, S), A \\ & \rightarrow eq(plus(pow(a, 2), pow(b, N), A \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), A \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), S) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), L) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), F) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), P(A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(S, A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(L, A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(V, A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(c, A) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(c, S)) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(c, N)) \\ & \rightarrow eq(plus(pow(a, 2), pow(b, 2), pow(c, 2)) \end{aligned}$$

2. számú melléklet:

A *test.xml* fájl forrása.

```
<?xml version="1.0" encoding="UTF-8"?>
<?oxygen RNGSchema="http://www.oxygenxml.com/docbook/xml/5.0/rng/dbmathml.rng" type="xml"?>
<article xmlns="http://docbook.org/ns/docbook">
  <title>Article Title</title>
  <sect1>
    <title>Section1 Title</title>
    <para>An equation:<emphasis role="italic">
      power(a,2)+power(b,2)=power(c,2)</emphasis></para>
    <para><inlineequation>
      <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
        power(a,2)+power(b,2)=power(c,2) </mml:math>
      </inlineequation></para>
    <para>This <emphasis role="italic">eq(frac(a,b),c)</emphasis> formula looks like
    this:<inlineequation>
      <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"> eq(frac(a,b),c)
      </mml:math>
    </inlineequation></para>
    <para>The next beauty:<emphasis role="italic">
      1+frac(root(3,2)+a,b)=4*power(c,5)+d</emphasis></para>
    <para><inlineequation>
      <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
        1+frac(root(3,2)+a,b)=4*power(c,5)+d </mml:math>
      </inlineequation></para>
    <para>The prefix formula of the following one is: <emphasis role="italic">
      >int(eq(x,0),2,power(x,2),x) </emphasis></para>
    <para><inlineequation>
      <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
        int(eq(x,0),2,power(x,2),x) </mml:math>
      </inlineequation></para>
  </sect1>
</article>
```

3. számú melléklet:

A *test.outputxml.xml* fájl forrása:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?oxygen RNGSchema="http://www.oxygenxml.com/docbook/xml/5.0/rng/dbmathml.rng" type="xml"?>

<article xmlns="http://docbook.org/ns/docbook">
  <title>Article Title</title>
  <sect1>
    <title>Section1 Title</title>
    <para>An equation:<emphasis role="italic">
      power(a,2)+power(b,2)=power(c,2)</emphasis>
    </para>
    <para>
      <inlineequation>
        <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
          <mml:mrow>
            <mml:mrow>
              <mml:msup>
                <mml:mrow>
                  <mml:mi>a</mml:mi>
                </mml:mrow>
                <mml:mrow>
                  <mml:mn>2</mml:mn>
                </mml:mrow>
              </mml:msup>
              <mml:mo>+</mml:mo>
              <mml:msup>
                <mml:mrow>
                  <mml:mi>b</mml:mi>
                </mml:mrow>
                <mml:mrow>
                  <mml:mn>2</mml:mn>
                </mml:mrow>
              </mml:msup>
            </mml:mrow>
            <mml:mo>=</mml:mo>
            <mml:msup>
              <mml:mrow>
                <mml:mi>c</mml:mi>
              </mml:mrow>
              <mml:mrow>
                <mml:mn>2</mml:mn>
              </mml:mrow>
            </mml:msup>
          </mml:mrow>
        </mml:math>
      </inlineequation>
    </para>
    <para>This <emphasis role="italic">eq(frac(a,b),c)</emphasis> formula looks like
    this:<inlineequation>
      <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
        <mml:mrow>
          <mml:mfrac>
            <mml:mrow>
              <mml:mi>a</mml:mi>
            </mml:mrow>
          </mml:mfrac>
        </mml:mrow>
      </mml:math>
    </inlineequation>
  </para>
</article>
```

```

        <mml:mrow>
            <mml:mi>b</mml:mi>
        </mml:mrow>
    </mml:mfrac>
    <mml:mo>=</mml:mo>
    <mml:mi>c</mml:mi>
</mml:mrow>
</mml:math>
</inlineequation>
</para>
<para>The next beauty:<emphasis role="italic">
    1+frac(root(3,2)+a,b)=4*power(c,5)+d</emphasis>
</para>
<para>
<inlineequation>
    <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
        <mml:mrow>
            <mml:mrow>
                <mml:mn>1</mml:mn>
                <mml:mo>+</mml:mo>
                <mml:mfrac>
                    <mml:mrow>
                        <mml:mrow>
                            <mml:mroot>
                                <mml:mrow>
                                    <mml:mn>3</mml:mn>
                                </mml:mrow>
                                <mml:mo>+</mml:mo>
                                <mml:mi>a</mml:mi>
                            </mml:mrow>
                        </mml:mrow>
                        <mml:mrow>
                            <mml:mi>b</mml:mi>
                        </mml:mrow>
                    </mml:mfrac>
                </mml:mrow>
                <mml:mo>=</mml:mo>
                <mml:mrow>
                    <mml:mrow>
                        <mml:mn>4</mml:mn>
                        <mml:mo>/</mml:mo>
                        <mml:msup>
                            <mml:mrow>
                                <mml:mi>c</mml:mi>
                            </mml:mrow>
                            <mml:mrow>
                                <mml:mn>5</mml:mn>
                            </mml:mrow>
                        </mml:msup>
                    </mml:mrow>
                    <mml:mo>+</mml:mo>
                    <mml:mi>d</mml:mi>
                </mml:mrow>
            </mml:mrow>
        </mml:math>

```

```

</mml:math>
  </inlineequation>
</para>
  <para>The prefix formula of the following one is: <emphasis role="italic">int(eq(x,0),2,power(x,2),x)
</emphasis>
</para>
  <para>
<inlineequation>
  <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
    <mml:mrow>
      <mml:munderover>
        <mml:mo>∫</mml:mo>
        <mml:mrow>
          <mml:mrow>
            <mml:mi>x</mml:mi>
            <mml:mo>=</mml:mo>
            <mml:mn>0</mml:mn>
          </mml:mrow>
        </mml:mrow>
      </mml:munderover>
      <mml:mrow>
        <mml:mn>2</mml:mn>
      </mml:mrow>
    </mml:mrow>
    <mml:msup>
      <mml:mrow>
        <mml:mi>x</mml:mi>
      </mml:mrow>
      <mml:mrow>
        <mml:mn>2</mml:mn>
      </mml:mrow>
    </mml:msup>
    <mml:mi>d</mml:mi>
    <mml:mrow>
      <mml:mi>x</mml:mi>
    </mml:mrow>
  </mml:mrow>
</mml:math>
  </inlineequation>
</para>
</sect1>
</article>

```