

SZÉCHENYI ISTVÁN EGYETEM
GÉPÉSZMÉRNÖKI, INFORMATIKAI ÉS VILLAMOSMÉRNÖKI KAR
MATEMATIKA ÉS SZÁMÍTÁSTUDOMÁNY TANSZÉK



DIPLOMAMUNKA

Véges automaták és reguláris nyelvek kapcsolata

Tóth Péter
Mérnök informatikus MSc

Győr, 2016

FELADATKIÍRÁS

Szakdolgozat címe: Véges automaták és reguláris nyelvek kapcsolata

Hallgató neve: Tóth Péter

Szak: Mérnök informatikus

Képzési szint: MSc

Típus: nyilvános

Feladat:

A formális nyelvek elméletének egyik alappillére az a fontos tétel, miszerint a reguláris kifejezésekkel megadható, a véges automatákkal felismerhető és a 3-típusú nyelvtannal generálható nyelvek osztálya megegyezik. A cél egy olyan szimuláció megvalósítása, amely bemutatja ezt a kapcsolatot (elsősorban a véges automatákkal felismerhető és a 3-típusú nyelvtannal generálható nyelvek között), igényes grafikai támogatással (látványos kimenet). A kezelendő nyelvtanok és automaták bonyolultságára a következő határok érvényesek: az automata állapotainak a száma max. 20; a kiküszöbölendő láncszabályok száma max. 10; a redukció során bevezetendő új nemterminálisok száma max. 20.

Részfeladatok:

a) A szakdolgozat elkészítéséhez szükséges résztvevő tevékenységek:

- A szükséges formális nyelvi háttér megismerése.
- A grafikai felület megtervezése és leprogramozása (automata megjelenítés, a felismerés és elfogadás bemutatása).
- A formális nyelvi modul leprogramozása.

b) A szakdolgozat főbb részei:

- A formális nyelvi háttér bemutatása.
- A formális nyelvi és a grafikai modul működésének megfelelően részletes leírása, dokumentáció.
- Kész automata ábrák.
- Nem kötelező részként: A levezetés folyamatának és az automata egyes lépéseinek animációs bemutatása. A reguláris kifejezésekkel való kapcsolat szemléltetése.

Belső konzulens: Dr. Kallós Gábor, docens, MSZT

Külső konzulens: Pukler Antal, novadat@kabelnet.hu

Győr, 2015. július 7.

tanszékvezető

belső konzulens

ÉRTÉKELŐ LAP

Szakdolgozat címe: Véges automaták és reguláris nyelvek kapcsolata

Hallgató neve: Tóth Péter

Szak: Mérnök informatikus

Képzési szint: MSc

Típus: nyilvános

A dolgozat beadható, nem adható be:

dátum

belső konzulens

A dolgozat bírálatra bocsátható, nem bocsátható:

dátum

tanszékvezető

A bíráló:

- Neve:
- Munkahelye:
- Beosztása:

A bíráló javaslata:

dátum

érdemjegy

bíráló vagy tszv.

A belső konzulens javaslata:

dátum

érdemjegy

belső konzulens

A ZVB döntése:

dátum

érdemjegy

ZVB elnök

NYILATKOZAT

Alulírott, Tóth Péter, mérnök informatikus, MSc szakos hallgató kijelentem, hogy a „Véges automaták és reguláris nyelvek kapcsolata” című diplomamunka feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével. Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelek.

dátum

hallgató

TÓTH PÉTER

**VÉGES AUTOMATÁK ÉS
REGULÁRIS NYELVEK KAPCSOLATA**

ÖSSZEFOGLALÓ

A formális nyelvek elméletének egyik alappillére az a fontos tétel, miszerint a reguláris kifejezésekkel megadható, a véges automatákkal felismerhető és a 3-típusú nyelvtannal generálható nyelvek osztálya megegyezik.

A dolgozat célja egy olyan szoftver megalkotása, amely bemutatja ezt a kapcsolatot (elsősorban a véges automatákkal felismerhető és a 3-típusú nyelvtannal generálható nyelvek között).

Az elkészült szoftverrel létrehozható nyelvtani szabályhalmaz, illetve megalkotható grafikusan egy véges automata, majd ezen két rendszer között tetszőlegesen konvertálható az adat. Tehát bármelyik változatban készült szabályrendszer átalakítható a másik fajta szabályrendszerré (a konvertáláshoz szükséges feltételeket a rendszer ellenőrzi).

A szoftver széles körben és akár oktatásban is való felhasználhatóságának érdekében webes rendszerként lett megalkotva és WIKI formátumú dokumentáció készül hozzá. Működéséhez nem szükséges szoftver vagy szoftverkomponens telepítése, mindössze a leggyakrabban használt böngészők egyike (pl.: Google Chrome) és egy élő internet kapcsolat.

Kulcsszavak: véges automata, reguláris nyelv, konvertálás

PÉTER TÓTH

**RELATIONSHIP BETWEEN FINITE
AUTOMATONS AND REGULAR LANGUAGES**

SUMMARY

There is an important thesis of the theory of the regular languages: when a language can be specified by regular expressions (REGEX), it can be recognized by finite automata and it can be generated by type-3 (regular) grammar (RG), then classes of these procedures are same.

The goal of this diploma work is to create a such software, which shows this relationship (mainly among those languages, which can be recognized by finite automata and can be generated by RG).

The completed program can create a set of rules of grammar or a finite automaton graphically as well, and data are freely convertible between these two systems. So any system of rules can be transformed into the other system of rules (The software checks conditions required for conversion).

This program has been designed as a Web application for the usability of software widely and even in education, and a WIKI format documentation was prepared to this. Software or software component installation does not require to run program, it needs only one of the most commonly used Web browsers (e.g. Google Chrome) and online Internet connection.

Keywords: finite automaton, regular language, conversion

TARTALOMJEGYZÉK

Bevezetés.....	1
1. A Fő modulok ismertetése.....	3
1.1. A reguláris nyelv modul.....	3
1.1.1. A terminális és nem terminális szimbólumok kijelölése.....	3
1.1.2. A szabály szerkesztése.....	4
1.1.3. A szabályhalmaz.....	5
1.1.4. A szabályhalmaz feldolgozásának műveletei.....	6
1.1.5. A szabályhalmaz feldolgozási műveletek beállításai.....	10
1.1.6. Szabályhalmaz eltávolítása.....	11
1.1.7. Eltárolt szabályhalmazok kezelése.....	11
1.2. A véges automata modul.....	12
1.2.1. A szerkesztő felület kezelése.....	12
1.2.2. Állapot mátrix.....	16
1.2.3. Egyéb funkciók.....	17
1.3. Dokumentáció menüpont.....	20
2. Reguláris nyelvi szabályokra alkalmazott eljárások ismertetése.....	21
2.1. Láncszabály mentesítés.....	21
2.1.1. Láncszabály mentesítés elmélete.....	21
2.1.2. Láncszabály mentesítés folyamata.....	21
2.1.3. Példa a szabály alkalmazására.....	22
2.2. Zsákutca mentesítés.....	22
2.2.1. Zsákutca mentesítés elmélete.....	22
2.2.2. Zsákutca mentesítés folyamata.....	23
2.2.3. Példa a szabály alkalmazására.....	23
2.3. Szabály hosszredukció.....	23
2.3.1. Szabály hosszredukció elmélete.....	24
2.3.2. Szabály hosszredukció folyamata.....	24
2.3.3. Példa a szabály alkalmazására.....	24
2.4. Üres szó lemma (λ szabályok eltávolítása).....	25
2.4.1. Üres szó lemma elmélete.....	25
2.4.2. Üres szó lemma algoritmus átalakítási folyamatának bemutatása.....	26
2.5. Az önmagukat tartalmazó szabályok eltávolítása.....	27
2.6. Szabály többszörözés eltávolítása.....	28
3. Konvertálási folyamat bemutatása egy példán keresztül.....	29
4. Rendszer információk.....	32
4.1. Több nyelvi rendszer felépítése.....	32
4.1.1. Nyelvi fájlok.....	32

4.1.2.Nyelv kiválasztása.....	32
4.1.3.Új nyelvi fájl bevezetése.....	32
4.1.4.A nyelvi fájlban található azonosítók rendszere.....	33
4.1.5.Szoftveres megvalósítás.....	33
4.2.Adatok mentése funkciók felépítése, szerkezete.....	34
4.2.1.Globális adatok.....	34
4.2.2.Reguláris nyelv modul által tárolt adatok.....	34
4.2.3.Véges automata modul által tárolt adatok.....	35
4.2.4.Szoftveres megvalósítás.....	36
5. Szoftver felépítése, környezete.....	37
5.1.Könyvtárszerkezet, fájlok.....	37
5.2.Képi anyagok.....	38
5.3.Szoftver megoldások.....	38
5.3.1.Paraméter átadás rendszere.....	39
5.3.2.Nyelvi szabályok reprezentációja.....	40
5.3.3.Grafikus felület összeállítása.....	41
5.3.4.Nyelvi szabályokat feldolgozó rutinok.....	44
5.3.5.JavaScript kiterjesztése.....	45
5.3.6.Egyéb.....	46
5.4.Szoftver tesztelése.....	48
5.4.1.Helyes működés tesztelés folyamata.....	48
5.4.2.Működés helyesség vizsgálatának eredménye.....	50
5.5.Szoftver telepítés.....	50
Irodalomjegyzék.....	51
Mellékletek.....	53

ÁBRAJEGYZÉK

1. Ábra: A T és a NT szimbólumok kijelölése.....	3
2. Ábra: Szabály szerkesztése.....	4
3. Ábra: A szabályhalmaz.....	5
4. Ábra: Szabályhalmaz feldolgozása.....	6
5. Ábra: Példa a konvertált automata formátumára.....	8
6. Ábra: Szabály konvertálása automatává.....	8
7. Ábra: Hiba szabály konvertálása alatt.....	8
8. Ábra: Szabályhalmaz által generált szavak (pl.).....	9
9. Ábra: Szabályhalmaz művelet beállításai.....	10
10. Ábra: Szabályhalmaz eltárolása.....	11
11. Ábra: Eltárolt szabályhalmazok kezelése.....	11
12. Ábra: A véges automata modul szerkesztő felülete.....	12
13. Ábra: Két állapot közötti átmenet.....	13
14. Ábra: Ugyanazon állapotra mutató átmenet objektum.....	13
15. Ábra: Átmenet objektum elfogatott állapotban.....	14
16. Ábra: Állapot objektum beállítható adatai.....	15
17. Ábra: Átmenet objektum beállítható adatai.....	16
18. Ábra: Állapot mátrix.....	16
19. Ábra: Automata vizualizáció eredménye.....	18
20. Ábra: Szó tesztelése automatánál.....	18
21. Ábra: Automata példa.....	19
22. Ábra: Automata nyelvtanná konvertálásának eredménye.....	19
23. Ábra: Automata eltárolása.....	20
24. Ábra: Eltárolt automata adatok kezelése.....	20
25. Ábra: Átalakítási sorrend.....	29
26. Ábra: Nyelvtanból konvertált automata.....	30
27. Ábra: Teszt automata kézi változata.....	48
28. Ábra: Teszt automata konvertált változata.....	49

TÁBLÁZATJEGYZÉK

1. Táblázat: Adatok mentése – Globális adatok.....	34
2. Táblázat: Adatok mentése – Reguláris nyelv modul.....	35
3. Táblázat: Adatok mentése – Véges automata modul.....	35
4. Táblázat: Adatok tárolásának kezelő rutinjai.....	36
5. Táblázat: A szoftver fájljai és használatuk.....	37

BEVEZETÉS

A dolgozat két nagy matematikai elmélet egymás közti kapcsolatát dolgozza fel.

A reguláris nyelvek a modern elméleti számítástudomány egyik fontos fogalmának, a formális módszereknek, azon belül a formális nyelvek matematikai elméletének a része. Ezen területen sokat köszönhetünk Noam Chomsky munkásságának.

Chomsky különböző kategóriákba sorolta a nyelvtanokat formai tulajdonságaik alapján:

Ha $G = (N, T, P, S)$ nyelvtan (grammatika), akkor

0-típusú (kifejezés struktúrájú, mondatszerkezetű), ha nincs rá vonatkozó külön korlátozás (Persze kell itt is: minden szabály baloldala tartalmazzon legalább egy nemterminális betűt)

1-típusú (környezetfüggő, CS), ha minden P -beli szabály $\beta A \gamma \rightarrow \beta \alpha \gamma$ alakú, ahol $\alpha \neq \lambda$ (azaz $\alpha \in (N \cup T)^+$; előfordulhat $\beta = \lambda$ és $\gamma = \lambda$); emellett megengedett az $S \rightarrow \lambda$ helyettesítés (azaz $\beta = \gamma = \alpha = \lambda$), de ha ilyen előfordul, akkor S nem lép fel egyetlen P -beli szabály jobb oldalán sem (Azaz az $A \rightarrow \alpha$ átírási szabály csak a $\beta\text{-}\gamma$ környezetben alkalmazható)

2-típusú (környezetfüggetlen, CF), ha P -ben $\forall$ szabály $A \rightarrow \alpha$ alakú (Azaz $N \rightarrow (N \cup T)^*$; a nemterminális itt mindig átírható)

3-típusú (reguláris, szabályos v. jobb lineáris), ha P -ben $\forall$ szabály $A \rightarrow xB$ vagy $A \rightarrow x$ alakú (Azaz a szabály jobb oldalán egy terminális jelekből álló szó áll úgy, hogy utána állhat még ...). [10]

Ebben a dolgozatban a 3-as típusú szabály van előtérben, hiszen ezt lehet átalakítani automatává. Ettől függetlenül a rendszer lehetőséget enged 2-es típusú szabályhalmaz megalkotására és rajta konverziós műveletek elvégzésére.

A másik nagy csoport a véges (vagy elfogadó) automaták a gépies bizonyításokkal kerültek előtérbe és velük kapcsolatos kutatások után kialakult az automaták matematikai elmélete.

Általánosságban az automatáról elmondható, hogy valamilyen absztrakt modell, amely mindig egy diszkrét állapotban van, és két állapot között valamilyen átmenet (pl. külső esemény) hatására lép át másik diszkrét állapotba. A véges jellege azért van, mert van neki egy output állapota, amikor a működése befejeződik. Az induló állapot általában fixen meghatározott (iniciális automata), de típustól függően tetszőleges kezdő állapotot is felvehet. Valamilyen szabályrendszer írja le a működését.

Az automatáknak még számos tulajdonsága lehet, de ennek ismertetése nem célja a dolgozatnak. Itt csak általánosságban kívántam megjeleníteni a két fő csoport alapvető tulajdonságait. A kettő közti konvertáláshoz szükséges folyamatokat és az átalakítás menetét a megfelelő szoftver részekben részletezem.

A szoftverben a két modul között lehet adatokat konvertálni a megfelelő szabályok betartása mellett, és ezzel gyakorlati bemutatásra kerül a két matematikai elmélet közti kapcsolat.

Az elkészített szoftver kezelése nem igényel komolyabb számítástechnikai előképzettséget, készítése során arra törekedtem, hogy egyszerűen kezelhető és egyértelműen megfogalmazott funkciói legyenek. A könnyebb kezelhetőség végett WIKI formátumú dokumentáció készült hozzá, amely részletesen bemutatja a szoftver használatát és funkcionalitásait. A WIKI dokumentációval két fontos célt értem el: a szoftver kezelésének segítése mellett tökéletes alapot nyújtott az ott megfogalmazott információ a diplomamunka dokumentációjának elkészítéséhez.

A nyelvi modulban sikeresen létrehoztam a tetszőleges sorrendben való különböző műveletvégzés rendszerét. Ez azért került előtérbe, mert egy másik szoftveres megvalósítás a témában a „Butterfly projekt” [5] csak fix sorrendű műveletvégzést engedélyez, és ez alapján mutatja be a nyelvi szabályhalmaz átalakítását. Ez viszont korlátozást jelent, és ebből jött az ötlet, hogy létre lehet-e hozni olyan rendszert, amelyben szabadon választható sorrendben lehet a szabályhalmaz átalakítását végrehajtani. Ezt sikerült megvalósítani. A szoftverben az átalakító eljárásokat tetszőleges sorrendben lehet lefuttatni. Azt sem korlátoztam, hogy egy átalakítási sorban csak egyszer lehessen egy átalakító eljárást rögzíteni, így egy feladat feldolgozási listába akár többször is felvehető ugyanaz az átalakító eljárás.

A feladat megfogalmazásában kiírt szép grafikai megjelenéshez a szoftverben készítettem egy egyedi grafikus szerkesztő felületet, amelyet teljes egészében magam terveztem és fejlesztettem. Sikerült megalkotni egy olyan automata szerkesztő felületet, amely kizárólag JavaScript kódból épül fel, és akár más célokra is fel lehet használni, mint pl. Petri-háló készítő program grafikai szerkesztő felülete.

A szoftver használata során nem küld információt a szerver felé. A felhasználó által készített nyelvi és automata adatok mindvégig a kliens gépen maradnak, ott kezelődnek és ott is mentődnek el. Ezzel a kialakítással teljesen védett a felhasználó és adatainak szuverenitása.

A szoftver webes elérhetősége: <http://tptg.hu/szakdoli>

1. A FŐ MODULOK ISMERTETÉSE

Ebben a fejezetben bemutatom a szoftver kezelését, és a megfelelő részekben röviden bemutatom a hozzájuk kapcsolódó elméleteket.

1.1. A reguláris nyelv modul

Ezen modulon belül tudunk reguláris nyelvi szabályrendszert létrehozni, és a létrehozott szabályrendszeren különböző konverziós műveleteket végrehajtani.

Megjegyzés: a rendszer megjegyzi az éppen szerkesztett szabályrendszer összes adatát, és egy újbóli lap betöltése vagy frissítés esetén megjeleníti azokat, így nem vesznek el az adatok!

1.1.1. A terminális és nem terminális szimbólumok kijelölése

A feladat kiírásakor peremfeltételként volt meghatározva, hogy először legyenek kijelölve a terminális (továbbiakban: T) és nem terminális (továbbiakban: NT) szimbólumok, majd csak utána lehessen szabályokat alkotni. Ezért a felső részben megjelenítem a lehetséges T és NT szimbólumokat. Ezzel korlátozzuk a szabályok létrehozásának lehetőségét. A T és NT sorokban található gombokra kattintva tudjuk kijelölni azt a szimbólumot, amelyet használni szeretnénk a szabály felépítésekor. A piros színnel jelölt szimbólumok a kiválasztott szimbólumok.

Az *S* jelű NT szimbólum automatikusan kijelölésre került, mivel az a kezdő szimbólum. És bár a kezdő szimbólum egyéb esetben tetszőlegesen választható az NT szimbólumok közül, ebben a feladatban ezt fixáltuk, és mindig kijelölt szimbólumként jelenik meg! Törlése nem lehetséges.

Reguláris nyelv modul

Terminális szimbólumok

a	b	c	d	e	f	g	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Nem terminális szimbólumok

A	B	C	D	E	F	G	H	J	K	M	N	P	Q	R	S	T	U	V	W	X	Y	Z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Kezdő szimbólum: S

1. Ábra: A T és a NT szimbólumok kijelölése

Az ABC betűinek használatát kínálok fel a választáskor. A T szimbólumok a kisbetűket, a NT szimbólumok a nagybetűket tartalmazzák. „A” karaktertől „Z” karakterig listázom az angol ABC szerint a betűket. A karakterekből eltávolítottam az „I”, „L” és „O” karaktereket, mivel megtévesztőek lehetnek és a T és a NT összekeverhetővé válhat egyes fontkészletek esetében. Ezzel a megszorítással 23 lehetőséget kínálok fel választásra az egyes csoportokban.

1.1.2. A szabály szerkesztése

A szabály szerkesztésében bevezettem bizonyos korlátozásokat:

1. Csak a korábban kijelölt N és a NT szimbólumokat lehet felhasználni szabály készítésekor.
2. A bal oldalon csak 1 darab NT szimbólumot lehet választani. Ezzel lekorlátoztam a létrehozható szabályok körét a Chomsky-féle 2-es és 3-as kategóriára.

2. Ábra: Szabály szerkesztése

Egy szabály elkészítésekor lehetőségünk van a legördülő választó mezőkből kiválasztani a szükséges T és NT szimbólumot. Ha a felső részben engedélyezzük a használatát, azaz kiválasztunk egy T vagy NT szimbólumot, az azonnal megjelenik a legördülő listában található szimbólumok között. Így bármikor változtathatunk a szükséges N és NT szimbólumok között.

Figyelem: ha viszont törölünk egy engedélyezést a T vagy NT szimbólumok között, akkor azonnal törlődnek azok a szabályok a szabály halmazból, amelyek tartalmazzák a most már nem engedélyezett T vagy NT szimbólumot! Ezzel a megszorítással mindig konzisztens marad a szimbólumok kijelölése a szabály halmazzal.

A szerkesztett szabály tagjainak számát lehet bővíteni vagy csökkenteni a „+” és „-” gombok segítségével. A minimális tagok száma egy. A maximális tagok száma nincs korlátozva.

Ha elkészültünk a szabály szerkesztésével, akkor a „Szabály felvétele a szabályhalmazba” gomb segítségével tudjuk rögzíteni az új szabályt.

Lehetőségünk van egy adott szabályt törölni a szabály halmazból a „Törlés” gomb segítségével. Mivel egy-egy szabály szerkezete viszonylag egyszerű, ezért nem láttam szükségesnek egy módosítás funkció létrehozását. Módosítás esetén kövessük a következő folyamatot: törlés, majd új szabály felvétele.

Az „Összes szabály törlése” gomb segítségével a teljes szabályhalmazt kitörölhetjük, és az elejéről kezdhetünk új szabályhalmazt építeni.

1.1.3. A szabályhalmaz

A szabályhalmaz blokkban jelenik meg az aktuálisan szerkesztett szabályhalmaz összes szabálya. Ebben a blokkban ugyanazt a szabályhalmazt 3 különböző formátumban mutatja meg a szoftver:

1. Külön szabályonkénti felsorolás (bal oldalt)
Ekkor minden egyes szabály külön sorban jelenik meg és ABC sorrendben.
2. NT-vel csoportosított formátum (jobb felül)
Ezen csoportban az egyes sorokban egy-egy NT-hez tartozó szabályok vannak felsorolva úgy, hogy az NT a sor elején található míg az egyes szabályok jobb oldalai „|” elválasztó karakterrel vannak felsorolva.
3. Generatív nyelvtani formátum (jobb alul)
A szabály formátumban 4 blokkban vannak elkülönítve az információk, balról jobbra haladva: NT felsorolása, T felsorolása, szabályok felsorolása, kezdő szimbólum megadása.

Jobb oldalt legalul megjelenik a Chomsky-féle szabály kategória értéke is tájékoztatásnak.

Szabályhalmaz

1. $A \rightarrow aB$	Törlés	Csoportosított leírás:
2. $A \rightarrow bB$	Törlés	
3. $B \rightarrow \lambda$	Törlés	$A \rightarrow aB \mid bB$
4. $C \rightarrow aA$	Törlés	$B \rightarrow \lambda$
5. $C \rightarrow bA$	Törlés	$C \rightarrow aA \mid bA$
6. $S \rightarrow aA$	Törlés	$S \rightarrow aA \mid bC$
7. $S \rightarrow bC$	Törlés	Nyelv leírás:

$G = (\{ A, B, C, S \} , \{ a, b \} , \{ A \rightarrow aB, A \rightarrow bB, B \rightarrow \lambda, C \rightarrow aA, C \rightarrow bA, S \rightarrow aA, S \rightarrow bC \} , S)$

Szabály kategóriája: 3

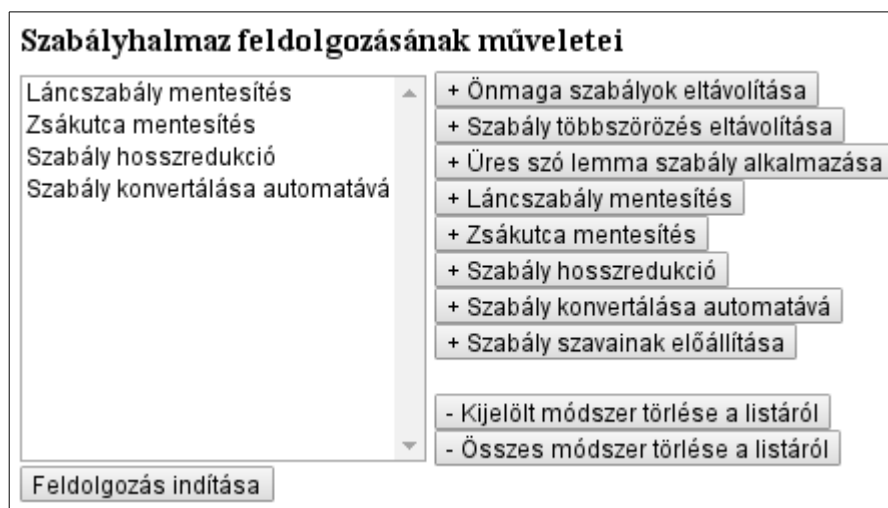
3. Ábra: A szabályhalmaz

1.1.4. A szabályhalmaz feldolgozásának műveletei

Ebben a blokkban tudjuk meghatározni azt, hogy a szabályhalmazon milyen átalakító algoritmusokat hajtsunk végre és azt milyen sorrendben tegyük meg. A rendszer azt biztosítja, hogy a kijelölhető átalakító algoritmusokat tetszőleges sorrendben és mennyiségben végre lehet hajtani. A következő algoritmusok hajthatók végre:

1. Önmaga szabályok eltávolítása
2. Szabálytöbbszörözés eltávolítása
3. Üres szó lemma szabály alkalmazása
4. Láncszabály mentesítés
5. Zsákutca mentesítés
6. Szabály hosszredukció
7. Szabály konvertálása automatává
8. Szabály szavainak előállítás

Ezek nagy része külön leírásban van részletezve, most az egyéb pontokat említjük csak.



4. Ábra: Szabályhalmaz feldolgozása

1.1.4.1. A szabály konvertálása automatává

Ez az a művelet, amely a feladat címében megfogalmazott egyik lehetséges utat biztosítja számunkra, miszerint a nyelvi szabályrendszer átalakítása véges automatává. Lényeges megemlíteni, hogy ez nem tehető meg minden szabályhalmazon. Számos szabályhalmazon szükséges egyéb konvertálási művelet végrehajtása előtt ezt a műveletet elhelyezzük a feladat listában. Ezek a konvertálási műveletek előkészítik a szabályhalmazt a megfelelő formátumra.

Ezen algoritmus kiválasztása a következő algoritmusok sorrendjében ajánlott:

1. Láncszabály mentesítés
2. Zsákutca mentesítés
3. Szabály hosszredukció
4. Szabály konvertálása automatává

A felsorolt szabályok készítik elő a nyelvi szabályhalmazt olyan állapotba, hogy átalakítható legyen automata szerkezetté. Az algoritmus csak akkor engedi az átalakítást, miután megvizsgálta, hogy minden egyéb feltételnek megfelel a szabályhalmaz. Az elfogadott szabály formátumok:

$$A \rightarrow aA$$

$$A \rightarrow a$$

$$A \rightarrow \lambda$$

Az $A \rightarrow aA$ formátumú szabály konvertálásakor két állapot objektumot készít a rendszer és összeköti őket egy darab átmenet objektummal. Természetesen ha a szabályban (mint a formátum példa mutatja) csak egy NT szimbólum szerepel, akkor csak egy állapot objektum van, és ekkor ugyanazon állapot objektumra mutató átmenet objektumot hoz létre.

Az $A \rightarrow \lambda$ típusú szabály egy elfogadó állapot objektumot készít.

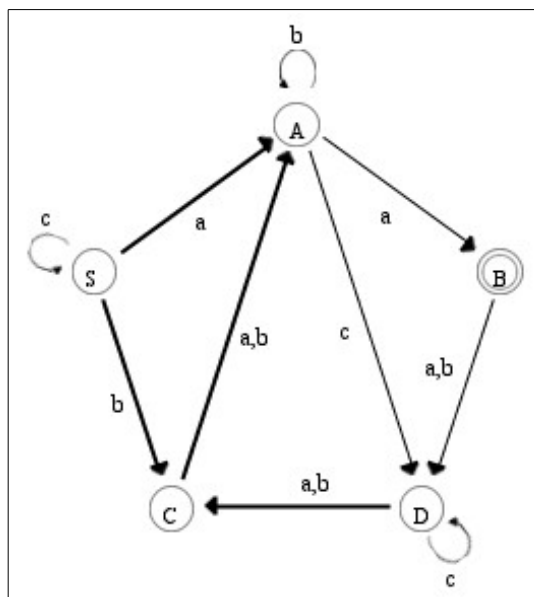
Az $A \rightarrow a$ formátumú szabály esetén szükség van egy kiegészítő $B \rightarrow \lambda$ szabályra is, azaz szükséges készíteni egy végállapot objektumot. Ekkor a szabály átalakul $A \rightarrow aB$ szabályra, és az $A \rightarrow aA$ formátumú szabály szerint képződik.

Az algoritmus végeredménye:

1. hibaüzenet: nem hajtható végre a konvertálás
2. konvertálás végrehajtott, át lehet lépni az automata modulra és megtekinthető a konvertált forma

Figyelem: Ezen pont alkalmazása sikeres végrehajtás esetén az aktuális végges automata adatainak törlésével/elveszésével jár!

Az következő ábrán található példából jól látható, hogy a szoftver egy kör kerületén helyezi el az állapot objektumokat, és az ugyanazon állapot objektumra mutató átmenet objektumokat sugár irányban kifelé forgatja el a jobb láthatóság érdekében. A két állapot objektum közötti átmenet objektumok felirata a nyíl irányának megfelelően jobb oldalra, hosszában középre van illesztve.



5. Ábra: Példa a konvertált automata formátumára

Sikeres konvertálás esetén nem lép át a szoftver automatikusan az automata modul felületére, viszont tájékoztat, hogy elkészült az automata képe és átléphetünk megtekinteni a formátumát.

Szabály konvertálása automatává

Az automata elkészült, lépjen át az automata modulhoz! [automata megtekintése](#)

6. Ábra: Szabály konvertálása automatává

Hibás adatok esetén a szoftver közli azt a szabályt, amely nem felel meg a konvertálási feltételeknek.

Szabály konvertálása automatává

A szabály halmaz tartalmaz olyan szabályt, amelyet nem lehet automatává alakítani! ($S \rightarrow AB$)

7. Ábra: Hiba szabály konvertálása alatt

1.1.4.2. Szabály szavainak előállítás

Lehetőség van arra, hogy megtekintsük a szabályhalmaz által generálható szavakat. Az előállításhoz több beállítás is tartozik, amelyeket a következő nagy pontban részletezek.

Nyelvtan által generált szavak:			
Szó	Előfordulás	Előállításra 1 db példa	
aa	1	$[S \rightarrow aA]$	$[aA, A \rightarrow aB]$ $[aaB, B \rightarrow \lambda]$ $[aa]$
ab	1	$[S \rightarrow aA]$	$[aA, A \rightarrow bB]$ $[abB, B \rightarrow \lambda]$ $[ab]$
baa	1	$[S \rightarrow bC]$	$[bC, C \rightarrow aA]$ $[baA, A \rightarrow aB]$ $[baaB, B \rightarrow \lambda]$ $[baa]$
bab	1	$[S \rightarrow bC]$	$[bC, C \rightarrow aA]$ $[baA, A \rightarrow bB]$ $[babB, B \rightarrow \lambda]$ $[bab]$
bba	1	$[S \rightarrow bC]$	$[bC, C \rightarrow bA]$ $[bbA, A \rightarrow aB]$ $[bbaB, B \rightarrow \lambda]$ $[bba]$
bbb	1	$[S \rightarrow bC]$	$[bC, C \rightarrow bA]$ $[bbA, A \rightarrow bB]$ $[bbbB, B \rightarrow \lambda]$ $[bbb]$

8. Ábra: Szabályhalmaz által generált szavak (pl.)

Figyelem: A szavak előállítása sokszor nagy számítási és idő igényű, ezért a beállításoknál csak akkor alkalmazzunk nagy értékeket, ha már megbizonyosodtunk, hogy kis értékek mellett előállíthatók a szavak, illetve figyeljünk az időkorlátozó beállításra. Túl nagy értékek esetén nagyon hosszú lefutási idővel lehet számolni!

Amennyiben valamely korlátozó feltétel miatt állt le a szavak előállítási folyamata (pl. elérte az 1000 db szó korlátot az előállított szavak száma), tehát a generálási folyamat meg lett szakítva, és ezáltal nem a teljes halmazt állította elő a rutin, akkor megjelenik egy figyelmeztető üzenet: „Nincs előállítva az összes lehetséges szó!”.

A „Max mélység” beállításának korlátozó feltétele miatt nem ad a szoftver figyelmeztető üzenetet! Tehát a generálható a^n jellegű szavak csak korlátozottan jelennek meg.

Azt is megtehetjük, hogy a listában minden egyéb átalakító algoritmus után elhelyezzük ez az algoritmust, és megfigyeljük hogyan alakulnak az előállított szavak az egyes algoritmusok lefutása után.

A szabály szavainak előállításakor nem csak a generált szavakat listázza a rendszer, hanem a szavak előállítási módját is megmutatja.

Pl.: $[S \rightarrow aA]$ $[aA, A \rightarrow aB]$ $[aaB, B \rightarrow \lambda]$ $[aa]$

A példában jól látható, hogy a vastag betűvel jelölt NT tag kifejtése történik a mellette szereplő nyelvtani szabály alkalmazásával. Az így kapott eredményt a következő blokkban láthatjuk. Az első blokk a kiinduló szabályt mutatja, az utolsó blokk pedig a létrehozott szót.

Az „Egyedi szavak megjelenítése” esetében az első megtalált lehetséges megoldást mutat meg, nem listázza az összes előállítási változatot. Ha szeretnénk megnézni az adott szóhoz tartozó összes lehetséges előállítási módot, akkor válasszuk a „Minden szó megjelenítése” módot, ekkor minden egyes előállított szó mellett ott van a módja is.

1.1.5. A szabályhalmaz feldolgozási műveletek beállításai

Az itt található beállítások befolyásolják a szabályhalmaz által generált szavak előállítását.

Szabályhalmaz feldolgozási műveletek beállításai

Max mélység: 10 ▼

Max szó: 1000 ▼

Max. futási idő: 10 sec

Szavak: Egyedi szavak megjelenítése ▼

9. Ábra: Szabályhalmaz művelet beállításai

Max. mélység

A „max. mélység” beállításával azt állíthatjuk be, hogy amikor a szabályhalmaz alkalmazásával a rendszer előállítja a lehetséges szavakat, akkor rekurzívan milyen mélységig kövesse a szabályokat a rendszer.

Pl. $A \rightarrow aA$ szabály esetében a végtelenségig lehetne a karaktert tartalmazó szavakat előállítani. Ezen beállítással ez lekorlátozható pl. maximum 10 karakter hosszú szóra.

Max. szó

A „max. szó” beállításával azt állíthatjuk be, hogy mennyi legyen a maximálisan előállított szavak száma. Az előző pontban is említett példa esetében ez végtelen lenne, ennek a beállításnak az alkalmazásával ez lekorlátozható pl. maximum 1000 db szóra.

Max. futási idő

Lehetőségünk van korlátozni a szavak előállításának futási idejét másodpercben megadva. Így megakadályozhatjuk, hogy a túl hosszú futási idő miatt „lefagyás” jelenséget okozzunk a böngészőben. A rendszer elég nagy maximális értéket enged beállítani, ami 86400 másodperc azaz 1 nap.

Szavak

A szavak előállításának végeredményét lehet szabályozni ezzel a kapcsolóval. A szavak megjelenési formátumát tudjuk szabályozni.

1. Egyedi szavak megjelenítése

A rendszer csoportosítja az egyes szavakat, és minden szót csak egyszer jelenít meg, valamint megmutatja a szó mellett a szó előfordulási mennyiségét is. A szavak ebben az esetben ABC sorrendben jelennek meg.

2. Minden szó megjelenítése

Minden egyes szót megjelenít a rendszer. Így előfordulhat, hogy egy szó többször is meg fog jelenni a listában. Ezzel a beállítással végig tudjuk követni a szavak előállítási sorrendjét (nincs sorba rendezés).

Amennyiben egy szó többször is szerepel a listában, akkor mellette megtekinthetjük a különböző előállítási módjait. Ez megmutatja, hogy egy nyelvtani szabály rendszer egy adott szót különböző úton (különböző szabályok egymás utáni alkalmazásával) képes előállítani.

1.1.6. Szabályhalmaz eltárolása

Lehetőségünk van eltárolni egy adott nyelvi szabályhalmazt. Meg kell adni egy elnevezést, és a funkció aktiválásával a „localStorage” rendszerben eltárolódik az aktuális szabályhalmaz összes adata.

10. Ábra: Szabályhalmaz eltárolása

1.1.7. Eltárolt szabályhalmazok kezelése

Az eltárolt szabályhalmazok listázódnak ki ezen menüpont alatt. Lehetőségünk van

1. Betöltés Az eltárolt adatok betöltésére
2. Felülírás Az eltárolt adatok felülírására az aktuális szabályhalmaz adataival
3. Törlés Az eltárolt adatok törlésére

A rendszer kiegészítésképpen megjegyzi a mentés dátumát és azt is kilistázza a felhasználó számára.

Eltárolt szabály halmazok		
Megnevezés	Mentés időpontja	Művelet
Példa 1	2016.04.15 21:59:24	Betöltés Felülírás Törlés

11. Ábra: Eltárolt szabályhalmazok kezelése

1.2. A véges automata modul

Ezen modulon belül tudunk elkészíteni egy véges automata modellt, és akár annak a működését szimuláltatni, tesztelni vagy át tudjuk konvertáltatni az elkészült automata szerkezetét nyelvi leírássá.

Megjegyzés: a rendszer megjegyzi az éppen szerkesztett automata adatait, és egy újbóli lap betöltése vagy frissítés esetén megjeleníti azokat, nem vesznek el az adatok!

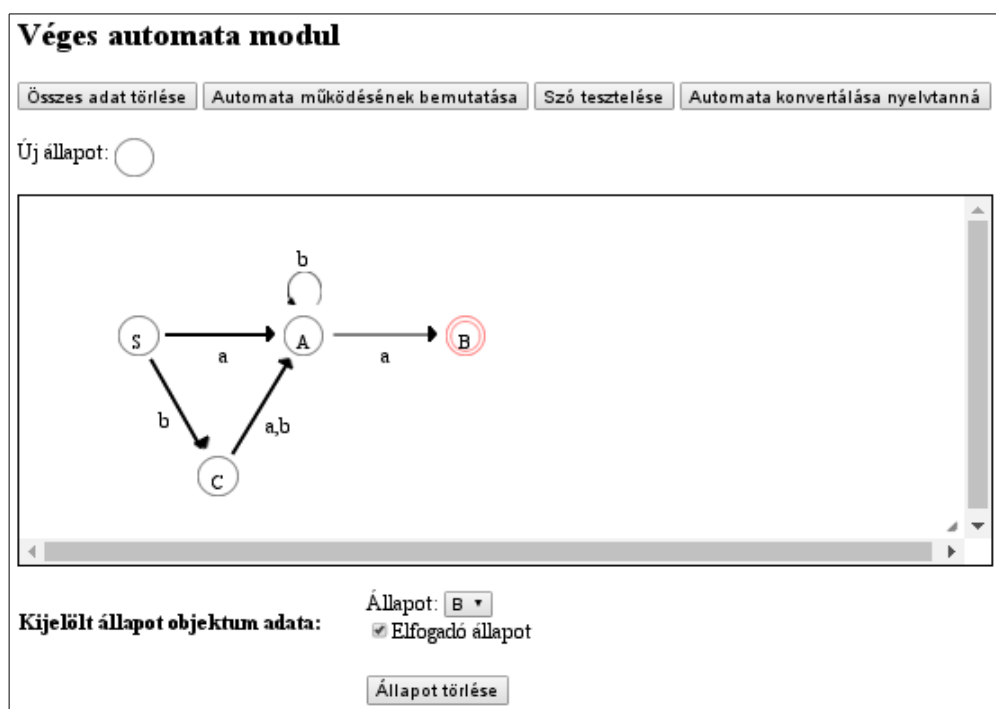
1.2.1. A szerkesztő felület kezelése

1.2.1.1. Szerkesztő ablak

Az automata összeállítására egy méretében módosítható szerkesztőfelület áll rendelkezésünkre. A szerkesztőfelület mérete a jobb alsó sarkában található kis háromszög alakú elem megfogásával/mozgatásával változtatható.

Ezen felületen belül helyezhetjük el az „állapot” objektumokat, és összeköttetéseket tudunk létesíteni az egyes objektumok között.

Megjegyzés: figyeljünk arra, hogy milyen böngészőt használunk, mert nem minden böngészőben működik a grafikus szerkesztő felület! A mellékletben információt kaphat a tesztelt és használható böngészőkről.



12. Ábra: A véges automata modul szerkesztő felülete

Új állapot objektum felvétele

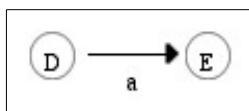
A szerkesztő ablak bal felső sarka felett látható az „Új állapot:” megnevezés és mellette egy üres állapot objektum (kör). Ezt az objektumot megfogva egér bal gombjának lefogásával tudjuk a szerkesztő felületre mozgatni. Ahol elengedjük az egér bal gombját ott elhelyeződik egy új állapot objektum.

Megfigyelhető, hogy elengedés után automatikusan értéket kap az új állapot objektum. Az első állapot objektum mindig a kezdő állapot jelét kapja, azaz az S betűt. Minden további állapot objektum az ABC betűit kapja sorban. A rendszer mindig ellenőrzi, hogy melyik az ABC-ben legelől lévő karakter szabad, és annak nevezi el az új objektumot. ($A..Z$, kivétel: I, L, O)

A kezdő állapot az az állapot, amelyből kiindulva feldolgozódik az automata. A rendszerben ez rögzítve lett az S betűre.

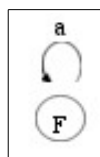
Új átmenet objektum felvétele

Két állapot objektum közé úgynevezett átmenet objektumot hozhatunk létre. Ezt úgy tudjuk végrehajtani, hogy ha az egeret ráhelyezzük egy állapot objektumra, akkor a felső részében megjelenik egy kék kis pötty. Helyezzük az egeret erre a kis kék pöttyre, és az egér bal gombját nyomva tartva már tudunk is egy nyilat mozgatni az állapot objektum körül. Mozgassuk az egeret ahhoz az állapot objektumhoz, amelyhez kapcsolatot szeretnénk teremteni, és engedjük el a bal egér gombot.



13. Ábra: Két állapot közötti átmenet

Megjegyzés: az elengedés lehet ugyanaz az objektum is, ahonnan elindítottuk a folyamatot! Ekkor ugyanazon állapotra mutató átmenet objektum jön létre.



14. Ábra: Ugyanazon állapotra mutató átmenet objektum

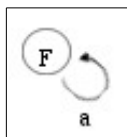
Sikertelen kapcsolatteremtés esetén nem születik kapcsolat két állapot objektum között. Ilyen lehet pl. ha már a 2 állapot objektum között volt előzőleg egy kapcsolat. Minden egyes két állapot objektum között csak egy átmenet kapcsolat hozható létre ugyanazon irányban!

Ha sikeres volt a folyamat, tehát egy kapcsolat létrejött, akkor megjelenik a két állapot objektum között az átmenet objektum (nyíl). Lényeges, hogy a nyíl mutatja a kapcsolat irányát. A rendszer automatikusan ellátja az átmenet objektumot az adott kiinduló állapot objektumnál szereplő következő átmenet szimbólummal. (*a..z*, kivétel: *i, l, o*)

Az átmenet objektumon szereplő szimbólumok megjelenítését (kiírás helyzete, pozíciója) teljesen automatikusan végzi a szoftver, mindig a nyíl irányában jobb oldalra, hosszában középre kerül a szimbólumok kirajzolása.

Ugyanazon állapotra mutató átmenet elforgatása

Lehetőségünk van arra, hogy egy olyan átmenetet hoztunk létre, amely önmaga állapot objektumra mutat. Ennek az átmenetnek a helyzetét az állapot objektum körül megváltoztathatjuk, így lehetőségünk van egy jobb megjelenítés létrehozására.



15. Ábra: Átmenet objektum elfogatott állapotban

Helyezzük az egeret az átmenet objektumra, aminek hatására megjelenik egy kis kék pötty. Ezután helyezzük az egeret a kék pöttyre, nyomjuk le és tartjuk nyomva a bal egérgombot, majd mozgassuk az egeret az objektum körül. Ennek hatására az átmenet objektum meg fogja változtatni helyzetét. Ha megfelelő irányba fordult az átmenet objektum, akkor engedjük el a bal egérgombot, és az átmenet objektum helyzete rögzítésre kerül.

Összes adat törlése

Az „Összes adat törlése” gomb segítségével az összes megszerkesztett objektum egyben törölhető, és tiszta „lappal” kezdhető a szerkesztés.

1.2.1.2. Kijelölt objektum kezelése

A szerkesztő felületen az állapot vagy az átmenet objektumra kattintva a szerkesztő ablak alatt megjelenik egy adat részletező blokk, ahol az adott objektum egyéb adatát tudjuk beállítani vagy pl. törölhetjük az objektumot.

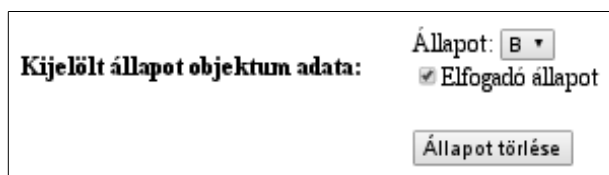
Figyelem: Minden változtatás az adat ablakban azonnal megváltoztat minden kapcsolódó adatot!

Állapot objektum

Állapot szimbólum meghatározása

Az állapot szimbólumokat a nagy „A” betűtől a nagy „Z” betűig lehet meghatározni. Az „P”, „L”, „O” szimbólumokat kivettük a listából biztonsági okokból (az összetéveszthetőség miatt).

A rendszer automatikusan azokat a karaktereket is kiveszi, amelyek már használva vannak a szerkesztő ablakban! Így minden állapot objektum csak egyszer szerepelhet a szerkesztő ablakban.



16. Ábra: Állapot objektum beállítható adatai

Az elfogadó állapot beállítása

Az elfogadó állapot bekapcsolását egy kipipálható beviteli mező bekapcsolásával tudjuk aktiválni.

Az elfogadó állapot azt jelenti, hogy ha az automata működését teszteljük, és a teszt során az elfogadó állapot jelöléssel ellátott állapot objektumot választjuk ki, akkor lezárhatjuk és konkretizálhatjuk az addig megfogalmazott szót (az automata működése leáll).

Állapot objektum törlése

Az „Állapot törlése” gomb hatására nem csak az adott állapot objektum törlődik, hanem a hozzá kapcsolódó összes átmenet objektum is!

Az átmenet objektum

Az átmenet szimbólum megadása

Az átmenet szimbólumokat a kis „a” betűtől a kis „z” betűig lehet meghatározni. Az „i”, „l”, „o” szimbólumok ki vannak véve a listából biztonsági okokból (az összetéveszthetőség miatt).

Egy adott átmenetre egyszerre több szimbólum is megadható. Ezt úgy tudjuk megtenni, hogy a „+” jelű gombbal egy új szimbólumot rendelhetünk az átmenet objektumhoz. Egy átmenet objektumra az elérhető maximális számú átmenet szimbólumot rendelhetünk azaz 23 darabot.

A „-” gombbal csökkenthetjük a szimbólumok számát.

Egy átmenet objektumon egy átmenet szimbólum csak egyszer szerepelhet! Ezt a feltételt automatikusan ellenőrzi a rendszer, és nem enged kiválasztani két azonos szimbólumot.

17. Ábra: Átmenet objektum beállítható adatai

Átmenet objektum törlése

Az „Átmenet törlése” gomb hatására csak az adott átmenet objektum törlődik!

1.2.2. Állapot mátrix

Az állapot mátrix megmutatja a szerkesztő ablakban létrehozott automata kapcsolati rendszerét táblázatos formában.

Állapot mátrix:	Állapot	Kezdő állapot	Elfogadó állapot	a	b
	S	X		A	C
	A			B	A
	B		X		
	C			A	A

18. Ábra: Állapot mátrix

A táblázat bal oldalán az állapot szimbólumok vannak felsorolva függőlegesen. A táblázat tetején pedig az átmenet szimbólumok vízszintesen.

A táblázat megmutatja, hogy a bal oldalon szereplő állapot szimbólum a felső részen található átmenet szimbólum hatására milyen másik állapot szimbólumba megy át.

Megjegyzés: Minden változtatás, amely az automata szerkezetében történik, azonnal frissítést eredményez az állapot mátrix szerkezetében is automatikusan!

1.2.3. Egyéb funkciók

1.2.3.1. Az automata működésének bemutatása

Lehetőségünk van arra, hogy teszteljük a létrehozott automata működését. Ez a teszt nem a teljes megoldást mutatja be, hanem csak egy véletlenszerűen létrehozott, egyetlen eredmény létrehozását mutatja meg.

Az elindítás után a szerkesztő ablak alatti adat blokkban tudjuk követni szövegesen a folyamatot, illetve ott tudjuk megnézni a létrehozott eredményt is.

1. Lépésszám
2. Létrehozott szó
3. Folyamat naplója

A bemutató folyamata:

1. Zöld színűre színeződik az az állapot, amely aktuálisan ki van választva.
2. Zöld színűre színeződnek azok az átmenet objektumok, amelyek a lehetséges továbbhaladási irányokat mutatják.
3. Vizualizációval piros színnel egymás után változtatódnak azok az átmenet szimbólumok és objektumok, amelyek közül egyet véletlenszerűen kiválaszt a rendszer.
4. Amikor megtörténik a kiválasztás, akkor az átmenet szimbólum és az objektum kék színűre vált.
5. Az előző pontban meghatározott átmenet kiválasztja a következő állapot objektumot.

A bemutató addig tart, ameddig:

1. elfogadó állapotba nem kerül a rendszer
2. hiba keletkezik a futás során, pl. olyan állapotba kerül a rendszer ami nem elfogadó állapot, és nincs belőle átmenet állapot létrehozva!

Automata működésének bemutatása:	Lépésszám: 3
	Készített szó: aa
	Napló:
	Állapot kijelölés: S
	Választható átmenetek: a,b
	Átmenet kiválasztásra került: a
	Állapot kijelölés: A
	Választható átmenetek: a,b
Átmenet kiválasztásra került: a	
Állapot kijelölés: B	
Automata elfogadó állapotba került!	

19. Ábra: Automata vizualizáció eredménye

1.2.3.2. Szó tesztelése

A funkcióval lehetőségünk van arra, hogy egy tetszőlegesen megadott szót teszteljünk az elkészített automatára. A funkció megadja számunkra, hogy az automata képes-e előállítani a megadott szót.

Eredmény lehet:

1. A megadott szó nem állítható elő az automatával!
2. A megadott szó elő állítható az automatával!

Szó előállításának vizsgálata:	Kérem adja meg a vizsgálandó szót!
	baba Vizsgálat indítása
	A vizsgálat sikeresen befejeződött!
	A megadott szó előállítható az automatával! (SCAAB)

20. Ábra: Szó tesztelése automatánál

Siker esetén a program megmutatja az előállítás sorrendjét is, azaz hogy az adott szót mely állapot objektumok elérésével lehet előállítani. A lehetséges megoldások közül az első megtalált megoldást mutatja meg.

1.2.3.3. Automata konvertálása nyelvtanná

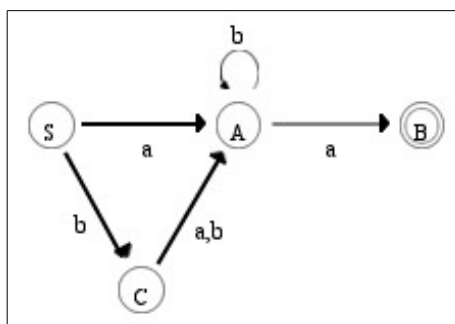
Ez a funkció szolgál arra, hogy a dolgozat címében megfogalmazott másik irányban való konvertálást bemutassa. Az elkészített automatából nyelvi szabály halmazt generál. A generálás után automatikusan átvált a képernyő a reguláris nyelvi modul ablakra.

Konvertálás szabályai:

1. Állapot objektum szimbóluma = NT szimbólum
2. Átmenet objektum egy-egy szimbóluma = egy-egy T szimbólum
3. Szabály felépítése: [kiinduló állapot objektum szimbóluma] = [átmenet objektum 1 db szimbóluma] [érkező állapot objektum szimbóluma]
4. Üres szabály (elfogadó állapottal bíró állapot objektumokra): [elfogadó állapot objektum szimbóluma] = λ

Az adatok konvertálása ebben az irányban egyszerűbb mint fordított helyzetben. Automatából minden esetben lehet nyelvi szabályhalmazt készíteni és nem szükséges az automata átalakítása vagy egyéb előfeldolgozása, mint a már korábban ismertett ellenkező irányban történő többlépcsős konvertálási folyamat.

Figyelem: A menüpont végrehajtása az aktuális nyelvi szabályhalmaz törlésével/elvesztésével jár!



21. Ábra: Automata példa

Szabályhalmaz		
1. $A \rightarrow aB$	Törlés	Csoportosított leírás:
2. $A \rightarrow bA$	Törlés	
3. $B \rightarrow \lambda$	Törlés	$A \rightarrow aB \mid bA$ $B \rightarrow \lambda$
4. $C \rightarrow aA$	Törlés	$C \rightarrow aA \mid bA$
5. $C \rightarrow bA$	Törlés	$S \rightarrow aA \mid bC$
6. $S \rightarrow aA$	Törlés	Nyelv leírás:
7. $S \rightarrow bC$	Törlés	
$G = (\{A,B,C,S\} , \{a,b\} , \{A \rightarrow aB, A \rightarrow bA, B \rightarrow \lambda, C \rightarrow aA, C \rightarrow bA, S \rightarrow aA, S \rightarrow bC\} , S)$		
Szabály kategóriája: 3		

22. Ábra: Automata nyelvtanná konvertálásának eredménye

1.2.3.4. A végges automata eltárolása

Lehetőségünk van eltárolni egy adott automata adatát. Meg kell adni egy elnevezést, és a funkció aktiválásával a „localStorage” rendszerben eltárolódik az automata adata.

Automata adatok eltárolása	
Mentési név:	Végges automata eltárolása

23. Ábra: Automata eltárolása

1.2.3.5. Eltárolt automata adatok kezelése

Az eltárolt automata adatok listázódnak ki ezen menüpont alatt. Lehetőségünk van

1. Betöltés Az eltárolt adatok betöltésére
2. Felülírás Az eltárolt adatok felülírására az aktuális automata adataival
3. Törlés Az eltárolt adatok törlésére

A rendszer kiegészítésképpen megjegyzi a mentés dátumát és azt is kilistázza a felhasználó számára.

Eltárolt automaták			
Megnevezés	Mentés időpontja	Művelet	
Példa automata	2016.04.18 11:54:39	Betöltés	Felülírás
		Törlés	

24. Ábra: Eltárolt automata adatok kezelése

1.3. Dokumentáció menüpont

A szoftver dokumentálása dokuwiki rendszerben történt. Ez azért kényelmes és praktikus, mert amellet, hogy többféleképpen felhasználható az ilyen dokumentáció a készítés során, számos előnye van egy sima szövegszerkesztőben készített anyaggal szemben, pl. minden egyes változtatás visszaneézhető az anyagban, és ezáltal egy véletlen és mentésre kerülő törlés visszaállítható könnyedén.

Emellet ugyan olyan funkcionalitással bír, mint egy szövegszerkesztőben készített dokumentáció, pl. szöveg mellett képek, táblázatok, stb. elhelyezhető az anyagban és a rendszer szabad keresést biztosít az egyes elemek szövegeire.

2. REGULÁRIS NYELVI SZABÁLYOKRA ALKALMAZOTT ELJÁRÁSOK ISMERTETÉSE

2.1. Láncszabály mentesítés

A láncszabály mentesítés végrehajtásához adjuk hozzá a „Feldolgozás műveletei” listához a „Láncszabály mentesítés” kapcsolót. A szabály végrehajtása a „feldolgozás” gomb megnyomására történik. A szabály lefutása során kiírja az indulási állapotot és a lefutás utáni eredmény állapotot.

2.1.1. Láncszabály mentesítés elmélete

Tetszőleges $G = (N, \Sigma, P, S)$ nyelvtan esetén az $A \rightarrow B$ alakú szabályokat láncszabályoknak (nemterminális átnevezés) nevezzük. Amennyiben P -ben nincsenek láncszabályok, akkor azt mondjuk, hogy G láncszabálymentes.

Segédállítás: Legyen $G = (N, \Sigma, P, S)$ környezetfüggetlen (2-típusú, CF) nyelvtan. Hozzá megadható olyan $G' = (N, \Sigma, P', S)$ láncszabálymentes CF nyelvtan, amelyre $L(G) = L(G')$. Továbbá, amennyiben G 3-típusú, akkor G' is 3-típusú lesz. [11]

2.1.2. Láncszabály mentesítés folyamata

Minden egyes szabályt végig kell vizsgálni, hogy van-e köztük olyan, amelynek bal és jobb oldalán is csak egy NT szimbólum található. Ha találtunk ilyen szabályt, akkor úgy alakítjuk át a szabályt, hogy a jobb oldalon található NT szimbólum összes szabályát létrehozzuk a bal oldalon szereplő NT-sal.

Figyelni kell arra, hogy ha a folyamat során önmagába forduló szabály jönne létre (pl. $A \rightarrow A$), azt mindig ki kell venni az eredményből!

Ezt a folyamatot addig hajtjuk végre, ameddig el nem fogynak az átalakítandó szabályok. A végrehajtás sorrendje nem számít mivel tetszőleges sorrendben végrehajtva a műveletet ugyanarra az eredményre fogunk jutni!

Megjegyzés: előfordulhat, hogy egy átalakítás eredményeként ugyanahhoz a bal oldali NT-hez egy másik, de szintén átalakítandó szabály jön létre. Természetesen akkor az így létrejövő új szabályon a műveletet végre kell hajtani. A pontos működés érdekében nem csak a még maradék szabályokat kell ellenőrizni, hanem az újonnan létrehozott szabályokat is!

2.1.3. Példa a szabály alkalmazására

Kiinduló szabály halmaz:

$$S \rightarrow B \mid Ca$$

$$B \rightarrow c \mid d$$

$$C \rightarrow aS$$

Megvizsgálva a szabályhalmazt, látható, hogy találunk egy olyan szabályt, amelynek mind a két oldalán csak 1-1 NT szimbólum található: $S \rightarrow B$.

Ezen szabályon végre tudjuk hajtani a láncszabály mentesítést.

Mivel a B NT szimbólumnak 2 darab szabálya is van ($B \rightarrow c$ és $B \rightarrow d$) ezért mind a két szabályt alkalmaznunk kell az S NT szimbólumon.

Tehát hozzuk létre a szükséges szabályokat: $S \rightarrow c$ és $S \rightarrow d$, így bővítve a szabály halmazunkat.

Eredmény szabály halmaz:

$$S \rightarrow c \mid d \mid Ca$$

$$B \rightarrow c \mid d$$

$$C \rightarrow aS$$

2.2. Zsákutca mentesítés

A szabály hosszredukció végrehajtásához adjuk hozzá a „Feldolgozás műveletei” listához a „Zsákutca mentesítés” kapcsolót. A szabály végrehajtása a „feldolgozás” gomb megnyomására történik. A szabály lefutása során kiírja az indulási állapotot és a lefutás utáni eredmény állapotot.

2.2.1. Zsákutca mentesítés elmélete

Azon nyelvtani jelek elhagyása, amelyekből nem vezethető le terminális szó. Meghatározzuk azon nyelvtani jelek $H \subseteq N$ halmazát, amelyekből levezethető valamilyen terminális szó.

A H_i halmaz olyan nyelvtani jelekből áll, melyekből közvetlenül levezethető $(H_{i-1} \cup T)^*$ -beli mondatforma, azaz legfeljebb i levezetési fázisban levezethető belőle terminális szó. A $H_1, H_2, \dots$ halmazsorozat a tartalmazásra nézve monoton növekvő, felülről korlátos halmazsorozat, így mindig stabilizálódik. Ha $H_{i+1} = H_i$, akkor legyen $H = H_i$.

A H_i halmaz ismeretében meghatározhatjuk

$$H_{i+1}\text{-t: } H_{i+1} = H_i \cup \{X \in N \mid \exists q : X \rightarrow q \in P, q \in (H_i \cup T)^*\}. \quad [8]$$

2.2.2. Zsákutca mentesítés folyamata

A zsákutca mentesítés folyamán a szabály halmazból eltávolítjuk azokat a szabályokat, amelyeket a kezdő állapotból kiindulva nem tudunk elérni, így nincsenek használva a feldolgozás során.

Figyelem: A következő egyszerűsített feltételezést nem jelenthetjük ki: van olyan NT szimbólum a szabályok között, amelyik egyik szabály jobb oldalán sem található. Ha találunk ilyen, akkor a hozzá tartozó összes szabályt törölhetjük.

Az előző mondatok azért nem állják meg a helyüket, illetve rossz zsákutca mentesítéshez vezetnének, mert ha van két NT szabály amelyek egymásra mutatnak, de nincsenek kapcsolatuk a kezdő szimbólummal, akkor azok az előző feltételezés szerint megmaradnának, de egy valóságos zsákutca mentesítés esetén mindkettőt törölni kell!

2.2.3. Példa a szabály alkalmazására

Kiinduló szabály halmaz ([9] alapján):

$$A \rightarrow Aa \mid b \mid a$$

$$B \rightarrow b \mid a$$

$$C \rightarrow a$$

$$S \rightarrow aSb \mid b \mid a$$

Az S szimbólumból kiindulva vizsgálva a felhasználható NT szimbólumokat megállapíthatjuk, hogy a $C \rightarrow a$ szabályt nem tudjuk alkalmazni, így ezt a szabályt törölhetjük a szabály halmazból.

Eredmény szabály halmaz:

$$A \rightarrow Aa \mid b \mid a$$

$$B \rightarrow b \mid a$$

$$S \rightarrow aSb \mid b \mid a$$

2.3. Szabály hosszredukció

A szabály hosszredukció végrehajtásához adjuk hozzá a „Feldolgozás műveletei” listához a „Szabály hosszredukció” elemet. A szabály végrehajtása a „feldolgozás” gomb megnyomására történik. A szabály lefutása során kiírja az indulási állapotot és a lefutás utáni eredmény állapotot.

2.3.1. Szabály hosszredukció elmélete

Tetszőleges $G = (N, \Sigma, P, S)$ nyelvtan esetén az $A \rightarrow B$ alakú szabályokat láncszabályoknak (nemterminális átnevezés) nevezzük. Amennyiben P -ben nincsenek láncszabályok, akkor azt mondjuk, hogy G láncszabálymentes.

Segédállítás 1: Legyen $G = (N, \Sigma, P, S)$ környezetfüggetlen (2-típusú, CF) nyelvtan. Hozzá megadható olyan $G' = (N, \Sigma, P', S)$ láncszabálymentes CF nyelvtan, amelyre $L(G) = L(G')$. Továbbá, amennyiben G 3-típusú, akkor G' is 3-típusú lesz.

Segédállítás 2: Legyen $G = (N, \Sigma, P, S)$ 3-típusú nyelvtan. Ekkor van olyan $G' = (N', \Sigma, P', S)$ 3-típusú nyelvtan, amelyre $L(G) = L(G')$, és P' -ben minden szabály $A \rightarrow aB$ vagy $A \rightarrow \lambda$ alakú, ahol $A, B \in N$ és $a \in \Sigma$. (Minden 3-típusú nyelvtanhoz megadható olyan szintén 3-típusú megfelelő, amelyre $L(G) = L(G')$, és P' -ben minden szabály $A \rightarrow aB$ vagy $A \rightarrow \lambda$ alakú). [11]

2.3.2. Szabály hosszredukció folyamata

A szabály hosszredukció algoritmus megkeresi az összes olyan szabályt, amelynek jobb oldalán minimum 3 tagból áll. Ezeket a szabályokat alakítja át úgy, hogy minden szabály amely a szabály halmazban található maximum 2 tagból álljon.

A megtalált minimum 3 tagból álló szabályokra először leellenőrzi, hogy valamely 2 tetszőleges tagját tudja-e helyettesíteni egy már meglévő olyan szabállyal, amely bal oldali NT tagjához csak pontosan egy azonos szabály tartozik. Ha nem talál ilyent, akkor bevezet egy új NT szabályt. A műveletet addig folytatja, amíg minden szabály maximum 2 tagból áll.

2.3.3. Példa a szabály alkalmazására

Átalakítandó szabály halmaz ([9] alapján):

$$A \rightarrow abB \mid a$$

$$B \rightarrow S \mid b$$

$$S \rightarrow \lambda \mid bB$$

$A \rightarrow abB$ szabály 3 tagú, tehát át kell alakítani!

Az ab tagok nem szerepelnek más szabályban sehol, így ezt nem lehet kiváltani.

A bB tagokhoz tartozik szabály, $S \rightarrow bB$, de mivel S -nek van másik szabálya is ($S \rightarrow \lambda$) ezért az S -t nem lehet használni szabály redukcióra!

Marad az új szabály bevezetése módszer. Az utolsó 2 tagot (bB) helyettesítjük és létrehozunk egy új szabályt. Bevezetjük C NT szimbólumot. Rögzítjük az új szabályt: $C \rightarrow bB$, majd helyettesítjük az $A \rightarrow abB$ szabályban a bB tagjait C -vel: $A \rightarrow aC$.

Átalakított szabály halmaz:

$$A \rightarrow aC \mid a$$

$$B \rightarrow S \mid b$$

$$S \rightarrow \lambda \mid bB$$

$$C \rightarrow bB$$

2.4. Üres szó lemma (λ szabályok eltávolítása)

A „Feldolgozás műveletei” listába fel tudjuk venni az „Üres szó lemma szabály alkalmazása” opciót, amely bekapcsolásával a létrehozott szabályhalmazt át tudjuk alakítani olyan szabályhalmazzá, amely maximum 1 db üres szó szabályt tartalmaz, mégpedig az $S \rightarrow \lambda$ szabály és ekvivalensen megegyezik az általa generált nyelv az eredeti szabályhalmaz által generált nyelvvel.

Megjegyzés: Ezen szabályról a fejlesztés során kiderült, hogy a dolgozat címében megfogalmazott konverzióhoz végül nem szükséges alkalmazni. A kifejlesztett modul benne van a programban, és általánosságban bemutatható a működése ezért jogosnak éreztem, hogy szerepeljen a dolgozatban.

2.4.1. Üres szó lemma elmélete

$G = (\{S, A, B\}, \{x, y\}, S, H)$, ahol H a következő szabályokból áll:

$$S \rightarrow AB, A \rightarrow BSB, A \rightarrow BB, B \rightarrow xAy, B \rightarrow \lambda, B \rightarrow x, B \rightarrow y.$$

$U_1 := \{B\}$ – azok a nemterminálisok, amelyekből λ közvetlenül megkapható.

$U_2 := \{A, B\}$ – azok a nemterminálisok, amelyekből U_1^* -beli szavak közvetlenül megkaphatóak!

$U_3 := \{S, A, B\}$ – azok a nemterminálisok, amelyekből U_2^* -beli szavak közvetlenül megkaphatóak.

Mivel U_3 a nyelvtan összes nemterminálisát tartalmazza, ezért $U := U_3 = U_4 = U_5 = \dots$

Tehát U -beli nemterminálisokból kapható meg az üres szó.

Mivel $S \in U$, ezért az üres szó benne van a G által generált nyelvben.

Készítsük el a $G' = (\{S', S, A, B\}, \{x, y\}, S', H')$ grammatikát, ahol H' a következő szabályokból áll:

1. $S' \rightarrow S, S' \rightarrow \lambda$. Ez a két szabály azért került be a H' szabályai közé, mert $\lambda \in L(G)$, ezért $\lambda \in L(G')$ -nek is teljesülnie kell!
2. Most vegyük azokat a H -beli szabályokat, amelyekben nem a λ szerepel a jobboldalon. Ezek a következők:

$$S \rightarrow AB, A \rightarrow BSB, A \rightarrow BB, B \rightarrow xAy, B \rightarrow x, B \rightarrow y.$$

3. És vegyük még azokat a szabályokat, amelyek H -ban nem szerepeltek, és amelyeket úgy kapunk, hogy H -beli szabályok jobb oldaláról az összes lehetséges módon elhagyjuk U halmaz nemterminálisait, figyelve azonban arra, hogy a jobb oldalon λ ne maradjon magában:

$$S \rightarrow A, S \rightarrow B, A \rightarrow BS, A \rightarrow SB, A \rightarrow B, A \rightarrow S, B \rightarrow xy. [9]$$

2.4.2. Üres szó lemma algoritmus átalakítási folyamatának bemutatása

Ha egy adott nyelvben szerepel egy olyan szabály, amely üres szó-t (λ) tartalmaz, akkor a nyelvi szabály halmazt át lehet alakítani úgy, hogy olyan szabályt kapjunk amely nem tartalmaz üres szót és ekvivalens az eredeti szabállyal. A következőben ismertetek egy algoritmust, amely a programban be van építve, és eltávolítja a nyelvi szabályhalmazból az üres szó szabályokat.

1. Első lépésben azonosítjuk azokat a szabályokat, amelyek tartalmaznak üres szót (λ). Ezeket elválasztjuk azoktól amelyek nem tartalmaznak üres szót. Pl.:

$$B \rightarrow \lambda$$

2. Az üres szót nem tartalmazó szabályok képezik az új szabály halmaz alapját. Pl.:

$$S \rightarrow Ba$$

ezeket a szabályokat kell kibővíteni az új szabályokkal.

3. Az azonosított szabályokból azokat amelyek $A \rightarrow \lambda$ formátumúak a rendelkezésre álló szabályokban behelyettesítjük mint üres karaktert (azaz töröljük a NT szimbólumot) és bővítjük az így keletkező új szabállyal a szabály halmazunkat. A fenti példák elvégezve a műveletet:

$$S \rightarrow Ba \quad \text{az eredeti szabályhalmaz}$$

$$S \rightarrow a \quad \text{átalakításkor keletkező új szabály}$$

szabályhalmazt képződik eredményül.

4. Ha egy szabályban több azonos NT szimbólum is szerepel, akkor külön-külön is elvégezzük a törlést és permutációban is. Pl.:

$$A \rightarrow \lambda$$

$$S \rightarrow AaA$$

szabályhalmaz esetén:

$$S \rightarrow aA$$

$$S \rightarrow Aa$$

$$S \rightarrow a$$

szabályhalmaz képződik.

5. Ha több nemterminálisnak is van λ szabálya, akkor az előző példához hasonlóan külön-külön is elvégezzük a helyettesítést és utána egymás kombinációjában is végrehajtjuk ugyanezt a műveletet. Pl.:

$$A \rightarrow \lambda$$

$$B \rightarrow \lambda$$

$$S \rightarrow AaB$$

szabályhalmaz esetén:

$$S \rightarrow aB$$

$$S \rightarrow Aa$$

$$S \rightarrow a$$

szabályhalmaz képződik.

6. Ha a szabályok feldolgozása közben az $S \rightarrow \lambda$ szabály előkerül valahol, akkor a kész szabályrendszerhez hozzá kell adni ezt a szabályt, mivel a nyelv által generált szavak között az üres szó-t is létre kell hozni, tehát szükség van erre a szabályra!

7. Az összes üres szó szabály feldolgozása után, ha az eredményhalmazban az átalakítást követően megint keletkezik üres szó szabály, akkor az algoritmust megint végrehajtjuk egészen addig, amíg nem kapunk olyan eredményt, amelyben már nincs üres szót tartalmazó szabály. Ilyen eset lehet pl.:

$$S \rightarrow Aa$$

$$A \rightarrow B \mid aB$$

$$B \rightarrow \lambda$$

szabályhalmaz feldolgozása (B behelyettesítése) után kapott szabályhalmaz:

$$S \rightarrow Aa$$

$$A \rightarrow \lambda \mid a$$

A keletkező szabályhalmazban megint van olyan szabály, amely λ tartalommal bír ezért az algoritmust újból lefuttatjuk a keletkezett szabály halmazon:

$$S \rightarrow a$$

$$A \rightarrow a$$

2.5. Az önmagukat tartalmazó szabályok eltávolítása

Az önmagukat tartalmazó szabályok eltávolítása szabály igen egyszerű. Minden szabály halmazban meg kell nézni, hogy van-e olyan szabály amelynek bal és jobb oldalán is ugyanaz a NT szimbólum található. Pl. $A \rightarrow A$

Egy ilyen szabály azt jelenti, hogy önmagát hozza létre a NT szimbólum. Ennek alapvetően nincs értelme, hasznossága, tehát ezek a szabályok mindig törölhetőek a szabályhalmazból. Általában ilyen probléma valamely szabály átalakításának eredményében keletkezik, így az eredmény halmazokon kell illetve lehet alkalmazni ezt a szabályt.

Megjegyzés: A szoftverbe beépített átalakító szabályok nagy része automatikusan elvégzi ezen szabály lefuttatását.

2.6. Szabály többszörözés eltávolítása

Ez a konvertálási mód egyszerűen annyit takar, hogy ha a szabályhalmazban kettő vagy több teljesen azonos szabály található, akkor a többszörözéseket egyszerűen el lehet távolítani a szabályhalmazból, úgy hogy minden szabály csak egyszer szerepeljen egy szabályhalmazban.

A művelet azért tehető meg, mivel semmilyen formában nem befolyásolja a szabályhalmaz által generált eredményt ha egy szabály többször is előfordul egy adott szabályhalmazban.

Megjegyzés: A szoftverbe beépített átalakító szabályok nagy része automatikusan elvégzi ezen szabály lefuttatását.

3. KONVERTÁLÁSI FOLYAMAT BEMUTATÁSA EGY PÉLDÁN KERESZTÜL

Ezen pontban bemutatom egy példán keresztül a reguláris nyelvtan átalakítási folyamatát végtes automata modellé. Készítettem egy szabályhalmazt (3-as típusút), amelyen keresztül bemutatható az összes szükséges átalakítási lépcső.

A létrehozott szabályhalmaz:

$$A \rightarrow D \mid aB$$

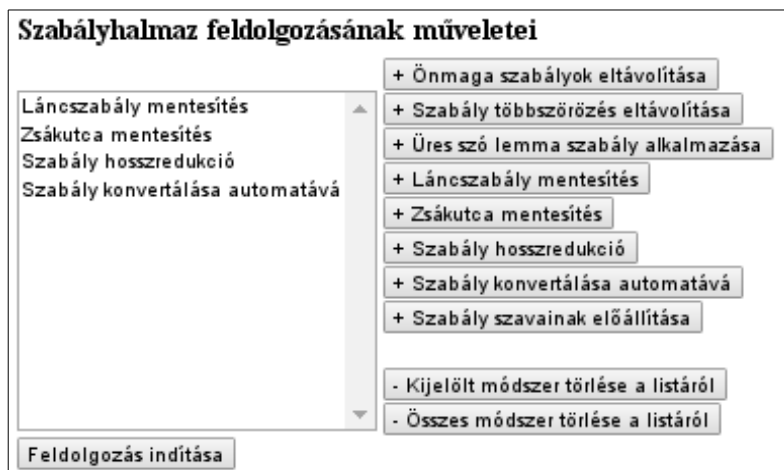
$$B \rightarrow \lambda$$

$$C \rightarrow D \mid aA$$

$$D \rightarrow bA$$

$$S \rightarrow abA \mid bC$$

Az első teendő, hogy be kell állítani a szükséges konvertálási lépcsőket a programban:



25. Ábra: Átalakítási sorrend

A feldolgozás elindításával első lépcsőben végrehajtódik a láncszabály mentesítés. A láncszabály mentesítéskor az $A \rightarrow D$ és $C \rightarrow D$ szabályok átalakulnak. Behelyettesítődik a $D \rightarrow bA$ szabály. Az így kapott $A \rightarrow bA$ és $C \rightarrow bA$ szabályokkal átalakítódik a szabályhalmazunk.

Láncszabály mentesítés eredményhalmaza:

$$A \rightarrow abB \mid bA$$

$$B \rightarrow \lambda$$

$$C \rightarrow abA \mid bA$$

$$D \rightarrow bA$$

$$S \rightarrow abA \mid bC$$

Második lépcsőben a kapott eredményhalmazon végrehajtódik a zsákutca mentesítés. Ekkor a $D \rightarrow bA$ szabály kidobódik a szabályhalmazból mivel S kezdő szimbólumból kiindulva nincs használatban.

Zsákutca mentesítés eredményhalmaza:

$$A \rightarrow aB \mid bA$$

$$B \rightarrow \lambda$$

$$C \rightarrow aA \mid bA$$

$$S \rightarrow abA \mid bC$$

Harmadik lépésben végrehajtódik a hosszredukció. A szabályhalmazunkban az $S \rightarrow abA$ szabályon kell alkalmazni a hosszredukció algoritmusát. Mivel nincs megfelelő helyettesítő szabály ezért újat kell bevezetni ($D \rightarrow bA$). Az új szabály módosítja a $S \rightarrow abA$ szabályt $S \rightarrow aD$ szabályra.

Szabály hosszredukció eredményhalmaza:

$$A \rightarrow aB \mid bA$$

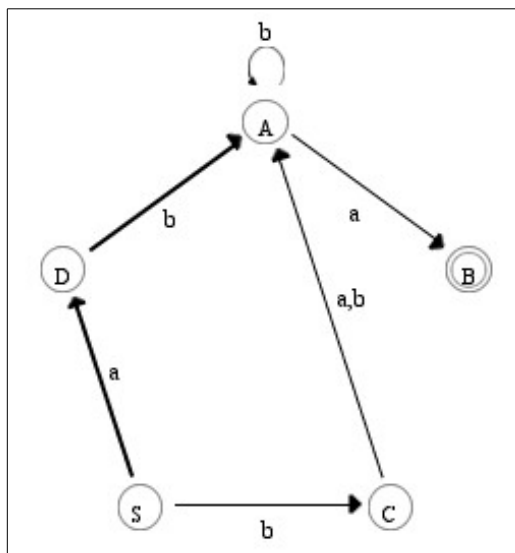
$$B \rightarrow \lambda$$

$$C \rightarrow aA \mid bA$$

$$S \rightarrow aD \mid bC$$

$$D \rightarrow bA$$

Ezzel el is készült a szabályhalmazunk arra a formátumra, hogy automatává konvertálható legyen. Így átlépve az automata modulba megtekinthetjük a kezdeti reguláris nyelvi szabályhalmazunkkal ekvivalens véges automata modellt.



26. Ábra: Nyelvtanból konvertált automata

Az átalakított automata és a szabályhalmaz összehasonlításából jól megfigyelhető, hogy minden szabályban található NT szimbólumból állapot objektum keletkezik, és a szabályban található T szimbólumból pedig átmenet objektum jön létre. Meg kell még említeni, hogy az üres szó szabályból elfogadó állapot jön létre.

Abban az esetben, amikor fordított átalakítást szeretnénk, azaz automatából akarunk nyelvtani szabályt készíteni, akkor az előzőekben leírt gondolatok fordítva is igazzá válnak. Tehát az állapot objektumból NT szimbólum, míg az átmenet objektumból T szimbólum keletkezik. És természetesen az is igaz, hogy minden elfogadó állapotból létre kell hozni egy üres szó szabályt.

4. RENDSZER INFORMÁCIÓK

4.1. Több nyelvű rendszer felépítése

4.1.1. Nyelvi fájlok

A rendszer úgy lett felépítve, hogy nincs „beégetve” egy meghatározott darabszámú nyelvhez tartozó szótár a szoftverbe, hanem szabadon alakítható és bővíthető a nyelvi fájlok készlete. A használt karakter kódolása UTF-8, amely lehetőséget kínál a Földön előforduló összes fontos nyelv kódolására.

A nyelvi fájlok a „language” könyvtárban kell elhelyezni, és „.lng” kiterjesztésűek. Mivel nem szükséges túl sok nyelvi szöveg (kb. 180 különböző szó vagy kifejezés) a rendszer használatához, ezért elég volt minden szükséges nyelvi anyagot egy fájlban elhelyezni.

Jelenleg elkészült nyelvi anyagok:

1. hu_hu.lng Magyar nyelv
2. en_gb.lng Angol nyelv
3. de_de.lng Német nyelv

4.1.2. Nyelv kiválasztása

Az egyes nyelvek kiválasztása a főmenü jobb oldalán található ország jelző zászlókkal lehetséges. A kiválasztást a rendszer megjegyzi, és a legközelebbi betöltéskor azzal a nyelvvel jelenik meg.

Azért, hogy olyan böngészőben is működjön a nyelv választás, amelyik nem rendelkezik a „localStorage” rendszerrel, kiegészítettem egy másodlagos módszerrel is ezt a funkciót. A zászlókra való klikkezéskor az oldal újratöltődik, és a cím sorban az „index.php” mellett paraméterként átadódik a kiválasztott nyelv azonosítója is, pl.: lang=hu_hu. Így képes a rendszer minden böngészőben a különböző nyelv kiválasztására.

4.1.3. Új nyelvi fájl bevezetése

A következőkben ismertetem az instrukciókat egy új nyelvre való lefordításhoz.

1. Másolja le valamelyik „.lng” állományt, amelyről le kívánja fordítani a rendszert.
2. A sor elején található azonosító számokat és az „=” jelet hagyja figyelmen kívül, és ne változtassa meg! A mögöttük lévő szöveget fordítsa le.
3. Közvetlenül az „=” mögé írja a lefordított szöveget egy sorban (ne legyen sortörés).
4. Ha esetleg van valamilyen HTML formázási rész a szövegben, akkor azt tartsa meg a megfelelő megjelenés végett.

Az új és elkészült nyelvi fájl megjelenítéséhez, rendszerhez való integrálásához kérje a rendszer fejlesztőjének segítségét.

4.1.4. A nyelvi fájlban található azonosítók rendszere

A nyelvi fájlban található azonosítók meghatározott rendszerbe vannak szervezve. Az első betűk jelzik, hogy melyik laphoz vagy funkcióhoz kapcsolódik az adott kifejezés.

m - Menü szavai

i - Impresszum oldal szavai

a - Véges automata modul szavai

n - Reguláris nyelv modul szavai

b - Véges automata modul-hoz tartozó script-ek által kezelt szavak

p - Reguláris nyelv modul-hoz tartozó script-ek által kezelt szavak

Az „m”, „i”, „a”, „n” betűk az egyes oldalak HTML kódjában az „LNG” nevű attribútummal adhatók meg, ezáltal egyszerűen bővíthetők egy későbbi fejlesztés során a szavak kezelése a HTML oldal szerkesztésével.

Ha egy nyelvi kód „t” karakterrel záródik, akkor nem a normál szöveget fogja lecserélni a rendszer, hanem az adott HTML taghoz tartozó TITLE attribútum szövegét, azaz a felbukkanó információs szöveget.

4.1.5. Szoftveres megvalósítás

A nyelvi fájlok betöltése nem a megszokott php-s háttér folyamatként történik, hanem minden nyelvi anyag cseréje online JavaScript-es rutinok által való cserét jelent. Az első lépésben AJAX hívással betöltődik a kiválasztott nyelvi fájl. Ezt az először anyagot soronként tömbbe rendezem (*aLang* globális változó). Ez a nyelvi adatbázis.

Két fontos kiszolgáló függvény van beépítve, amely a nyelvi szövegek cseréjét vagy kiszolgálását hajtják végre.

A *funcGetLang(s, p1, p2)* függvény segítségével lehet lekérdezni egy adott nyelvi azonosítóhoz tartozó szöveget. A függvénynek 3 paramétere van.

Az elsővel (*s*) megadhatjuk a nyelvi azonosítót, amelyhez tartozó szöveget keressük. A következő kettővel (*p1, p2*) pedig megadhatunk olyan beágyazott szövegeket, amelyeket ki kell cserélni a szövegrészben. A *p1*-et *%s*-sel a *p2*-t *%t*-vel lehet jelölni a szövegben. Egy példán szemléltetve, a nyelvi fájlban a következő szöveg található az azonosítójával együtt:

b10=Választható átmenetek: %s

A *%s* olyan szövegrész, amely futás közben keletkezik, előre nem definiálható. Jelen példaszöveg az automata bemutatója közben van használva és azt mutatja meg milyen lehetséges átmeneteket lehet választani az adott állapot objektumból kiindulva. Ez az érték minden egyes átmenet objektumnál más és más lehet. Ezzel a módszerrel dinamikus szövegeket lehet létrehozni és cserélni. PHP-ban erre *sprintf()* függvényt használunk

általában, de JavaScript-ben külön meg kellett valósítani. A *%s* és *%t* paramétereket a szövegben tetszőleges helyre és sorrendben lehet elhelyezni.

Ha olyan azonosítót adunk meg a rutinnak, amely nem található a betöltött nyelvi adatbázisban akkor *false* értékkel tér vissza a függvény.

A *funcLangFeltolt()* függvény arra szolgál, hogy egy betöltött HTML oldal kódjában kicserélje a szükséges szövegrészeket. Ezáltal ez az egy függvény egyben feliratozza a teljes betöltött új modult. A 4.1.4-es pontban említettek szerint az összes megjelölt HTML tagot megkeresi. Természetesen csak a szükséges megjelölt *sAzon* kezdőbetűvel rendelkező azonosítókat gyűjti ki.

```
$("[lng]").each(function() {
    sz=new String($(this).attr("lng"));
    if(sz.charAt(0)==sAzon) a.push(sz);
});
```

Miután ezeket összegyűjtötte, utána beállítja a felelhető összes szövegrészt. Mind a normál szöveget, mind a *TITLE*-ben tárolt feliratot. A nyelvi anyag kikereséséhez az előzőekben tárgyalt *funcGetLang()* függvényt használja fel.

4.2. Adatok mentése funkciók felépítése, szerkezete

Az összes adat tárolása kliens oldali HTML5-ös „localStorage” tároló rendszerben történik. Így semmilyen adat nem jut el a szerver felé a felhasználó által mentett adatok közül.

4.2.1. Globális adatok

A következő táblázatban bemutatom az egész rendszerre vonatkozó beállítási adatokat.

1. Táblázat: Adatok mentése – Globális adatok

Azonosító	Leírása
lang	Kiválasztott nyelv azonosítója, pl.: hu_hu
menu	Kiválasztott menüpont azonosítója, pl.: nyelvtan

4.2.2. Reguláris nyelv modul által tárolt adatok

A következő táblázatban bemutatom a reguláris nyelvi modulban található beállítási adatokat. A mentés sorszáma 1-től folyamatosan növekvő érték. A szoftver automatikusan megkeresi a lehető legkisebb használható értéket egy új mentés alkalmával.

[idx] – Mentés sorszáma

[sorsz] – Szabály sorszáma

2. Táblázat: Adatok mentése – Reguláris nyelv modul

Azonosító	Leírása
shd[idx]	Mentés dátuma
shm[idx]	Mentés megnevezése
sht[idx]	Terminális karakterek „,”-vel elválasztva, pl.: a,b,c
shn[idx]	Nem terminális karakterek „,”-vel elválasztva, pl.: A,B,S
shs[idx]_[sorsz]	Szabályok felsorolva, pl.: A=aB
maxiteracio	A szó előállításához használt rutin maximális rekurzív mélysége
maxszó	Maximálisan előállított szavak száma (db)
maxtime	Maximum futási idő másodpercben megadva
osszszó	A szavak előállításának végeredménye milyen legyen 0 – Egyedi szavak megjelenítése 1 – Minden szó megjelenítése

4.2.3. Véges automata modul által tárolt adatok

A következő táblázatban bemutatom a véges automata modulban eltárolt beállítási adatokat. A mentés sorszáma 1-től folyamatosan növekvő érték. A szoftver automatikusan megkeresi a lehető legkisebb használható értéket egy új mentés alkalmával.

[idx] – Mentés sorszáma

[sorsz] – Objektum sorszáma

3. Táblázat: Adatok mentése – Véges automata modul

Azonosító	Leírása
asd[idx]	Mentés dátuma
asm[idx]	Mentés megnevezése
asa[idx]_[sorsz]	Állapot adatok (körök) Részadatok „,”-vel elválasztva: 1.: Állapot objektum sorszáma 2.: Bal felső sarok left koordináta 3.: Bal felső sarok top koordináta 4.: Állapot szimbólum (A-Z) 5.: Vég állapot-e? (0-Nem; 1-Igen)

Azonosító	Leírása
asn[idx]_[sorsz]	Átmenet adatok (nyilak) Részadatok „,”-vel elválasztva: 1.: Átmenet objektum sorszáma 2.: Kiinduló állapot objektum sorszáma 3.: Mutatott állapot objektum sorszáma 4.: Forgatási szög (A-A típusúnál használva, fokban megadva, észak=0 fok, jobbra fordulva pozitív értékkel) 5-n.: Állapot szimbólumok (a-z, több is lehet)

4.2.4. Szoftveres megvalósítás

Mivel 2 teljesen különböző adatszerkezetről van szó a két főmodult nézve ezért mindkét modulnál külön meg kellett valósítani a mentés-visszatöltés funkciót, és nem lehetett egy azonos modullal megoldani.

4. Táblázat: Adatok tárolásának kezelő rutinjai

Funkció	Nyelvi modul	Automata modul
Mentés törlése	<i>funcSzabalyHalmazMentesTorlese()</i>	<i>funcAutomataMentesTorlese()</i>
Adatok mentése	<i>funcSzabalyHalmazSave()</i>	<i>funcAutomataSave()</i>
Adatok betöltése	<i>funcLoadSzabalyHalmaz()</i>	<i>funcLoadAutomata()</i>
Mentések listázása	<i>funcGetSzabalyHalmazSaveList()</i>	<i>funcGetAutomataSaveList()</i>

Jól látható az előző táblázatból, hogy egységes szerkezet kialakítására törekedtem és figyeltem az olvashatóságra. A többi kezelő függvénynél is ebben a szellemben jártam el.

A rutinok megalkotásánál egy nagyon fontos részletre kellett figyelni, hogy egy adat mentése nem egy adatot jelent, hanem egy adat csoportot. Például a szabály halmaz mentése több különálló szabály eltárolását jelenti. A fenti táblázatokban részleteztem milyen azonosítók tartoznak egy csoportba. Mint mentéskor, mint törléskor a teljes csoportot kell kezelni és egyben kell menteni az összes tagját illetve egyben kell törölni az összes elemet.

Figyelni kellett arra is, hogy ha a felhasználó a felülírás funkciót választja ki, akkor létezhet olyan eset, hogy a már mentett anyag nagyobb, mint a most mentendő anyag. Így a tényleges adat elmentése után a fennmaradó felesleges adatokat el kell távolítani a már meglévő mentésből.

5. SZOFTVER FELÉPÍTÉSE, KÖRNYEZETE

A szoftver nem igényel speciális beállításokat, alap webszerveren elfuttatható. Tárterület igénye alig haladja meg az 500 kB-ot. Működtetéshez szükséges a web kiszolgáló rendszer mivel tartalmaz php kódrészleteket, így statikus weboldalként nem működik.

A grafikus elemek CSS3-as formázásokat tartalmaznak, amelyeket sajnos nem minden böngésző tud jelenleg tökéletesen megjeleníteni. Tesztelés során Google Chrome, Opera és Safari böngészőknél találtam teljesen működésképesnek az oldalt. Ezért ezeket a böngészőket ajánlom, amelyre a rendszer figyelmezteti is a felhasználót.

Az adatok mentéséhez kliens oldali HTML5-ös „localStorage” adattároló rendszert használ a web oldal. A tesztek során kiderült, hogy ennek a tárolási módszernek már széles körű támogatottsága van. Ennek ellenére a rendszer ellenőrzi az adattároló motor működésének lehetőségét, és ha nem találja, akkor figyelmezteti a felhasználót erre, és nem engedi használni az olyan modulokat, funkciókat, amelyekhez erre lenne szükség.

Az oldal nem használ cookie adatokat. Így nem szükséges, és nem is építettem be a törvényileg előírt figyelmeztető és beleegyező nyilatkozat megjelenítését.

5.1. Könyvtárszerkezet, fájlok

A következő táblázatban megtekinthető az összes a rendszerhez szükséges könyvtár és fájl és használatuk leírása.

5. Táblázat: A szoftver fájlljai és használatuk

Könyvtár/File	Leírás
index.php	Kezelő keretprogram A főmenüt és head rész beállításait tartalmazza
favicon.ico	Program ikonja
include	Kezelő felületek könyvtára Ezek a fájlok AJAX hívással vannak betöltve a <i>#munkaterulet</i> blokkba
include/automata.php	Véges automata modul felülete A véges automata modul HTML kódja, a menüpont aktiválásakor töltődik be
include/impresszum.php	Impresszum menüpont felülete A impresszum oldal HTML kódja, a menüpont aktiválásakor töltődik be

Könyvtár/File	Leírás
include/nyelvtan.php	Reguláris nyelv modul felülete A nyelvi modul HTML kódja, a menüpont aktiválásakor töltődik be
js	JavaScript rutinok könyvtára
js/index1.js	A rendszer kezelő JavaScript függvényei Ebben az egy fájlban van elhelyezve az összes JavaScript kód
js/jquery.js	JQuery segéd függvénytár
js/jqueryui.js	JQueryUI segéd függvénytár
language	Nyelvi fájlok könyvtára
language/hu_hu.lng	Magyar nyelvi fájl
language/en_gb.lng	Angol nyelvi fájl
language/de_de.lng	Német nyelvi fájl
style	Formázási fájlok könyvtára
style/style1.css	A rendszer formázási fájlja Az összes formázási parancs ebben az egy fájlban van elhelyezve
style/smoothness	A JQueryUI formázási könyvtára
picture	A képi elemek könyvtára A mellékletben megtalálható a pontos tartalma leírással együtt

5.2. Képi anyagok

Az szoftverben található összes képet én szerkesztettem GIMP szoftverrel a megfelelő formátumra. Van köztük, amely teljesen saját készítésűek és vannak, amelyek valamely alapanyagból kiindulva alakítottam a megfelelő formátumra.

A képi anyagok kinézete és használatuk leírása a mellékletben megtekinthetőek (alább).

5.3. Szoftver megoldások

Ebben a részben ismertetek pár olyan megoldást, amelyeket a szoftver tartalmaz, és érdekesebbnek ítélem megemlíteni a fejlesztésből. Itt jegyzem meg, hogy a teljes szoftver fel van kommentálva, így a forráslistát olvasva is megérthető egy szakember számára a szoftver működése.

5.3.1. Paraméter átadás rendszere

Ezt azért emelem ki, mert ennek köszönhető elsősorban, hogy meg tudtam oldani a tetszőleges sorrendben történő adatfeldolgozás rendszerét. A megfelelően szabályozott és szabványszerűen kezelt paraméter átadás biztosítja azt, hogy az egyes konvertáló rutinok egymással kompatibilisek lesznek, és bármelyik rutin képes fogadni a másik rutin kimenetét bemenetként. Ezzel biztosítva egymás után fűzésüknek tetszőleges sorrendjét.

A JavaScript rutinyűjteményben megfigyelhető, hogy a legtöbb funkció egyetlen paramétert tartalmaz, melynek neve: „param”. Ez a változó nem egy megszokott „általános típussal” rendelkezik, hanem JSON adatszerkezetben várja az adatokat. A JSON adatszerkezet kellő rugalmasságot biztosít arra, hogy korlátozás nélküli paramétert és változó típust lehessen átadni a meghívó függvényről a meghívott függvénynek. Nagy előnye még, hogy visszatérési értéként is ugyanilyen szabadsággal használható. Alap esetben visszatérő értéknek egy változó értékét lehet átadni. Ezzel a módszerrel viszont több különböző változót és értékét lehetett átadni egyik függvényről a másiknak.

Mivel csak egy paraméter van a függvény definícióban, ezért még nagyobb szerep jut a megfelelő dokumentálásnak! Minden egyes ilyen rutin előtt megtalálható a pontos leírás arra vonatkozólag, hogy milyen paramétereket és értékeket vár az adott rutin a JSON adatszerkezetben. Ez a dokumentáció nem csak a bemeneti paramétereket tartalmazza, hanem a visszaadott paraméter szerkezetét is.

A következő forráslista részlet jól szemlélteti a függvény előtt megtalálható paraméter leírást:

```
/*
-----
Szabályból T és NT halmaz megállapítása
-----
Paraméterek: (param - Paraméterek átadása JSON formátumban)
.show      - Ki legyen-e íratva a megállapított halmaz (0-nem; 1-igen)
.aSzabaly  - Szabály halmaz tömbje
-----
Kimenet:
.error     - Sikeres volt-e a folyamat
.aTerm     - Terminális halmaz tömbje
.aNoTerm   - Nemterminális halmaz tömbje
----- */
function funcGetTNTfromSzabaly(param){ ... }
```

A JavaScript lehetőséget kínál arra is, hogy egyes paraméterek ne értéket vegyenek fel, hanem egyszerűen a paraméter megléte vagy nemléte hordozzon információtartalmat függvény hívásakor. Ezt a módszert akkor használom, ha valamilyen bináris információt akarok átadni egyik függvényről a másiknak. Általában akkor használom, ha viszonylag ritkán kell egy paramétert átadni, így nem kell minden meghíváskor definiálni ezt csak akkor amikor pl. tiltani akarok valamit. Ilyen megoldás a „noaddvalasz” paraméter, amely megtiltja az adott rutinnak, hogy a válaszlista közé naplózza az eredményt. Ezt nem kell sűrűn kiadni hiszen az eredményt általában naplózni kell.

A JSON adatszerkezet általános felépítés:

{[1. változó neve]: [1. változó értéke], [2. változó neve]: [2. változó értéke], ...}

Egy darab paraméterben való több paraméter átadásának kötelező hozadéka, hogy rutin indulásakor és rutin hívás visszatérésakor először a paramétert fel kell dolgozni. Minden egyes ilyen rutin úgy kezdődik, hogy a bejövő paramétereket megfelelőképpen szortírozza, ellenőrzi. Pl.:

```
var b=param.show==1;
var idx=new Number(param.index)
var aSzabaly=param.aSzabaly.clone();
```

A visszatérő értéknél hasonló módszerrel történik a feldolgozás. Pl.:

```
av=funcGetTNTfromSzabaly({show:0,aSzabaly:aSzabaly});
if(av.error)return {error:true};
var aTerm=av.aTerm.clone();
var aNoTerm=av.aNoTerm.clone();
```

Van egy kötelező paraméter visszatérő érték esetén, ez az „error” paraméter. A paraméter megmutatja, hogy sikeres volt-e a függvény lefutása vagy történt-e valamilyen hiba. Siker esetén *false* értéket vesz fel.

5.3.2. Nyelvi szabályok reprezentációja

Szükséges volt kitalálni valamilyen módszer arra, hogyan lehet egy nyelvi szabályt illetve magát a nyelvi szabály halmazt tárolni és rajta műveleteket elvégezni. Viszonylag leegyszerűsítette ezt az a tény, hogy le lett korlátozva a használt T és NT szimbólumok száma az elérhető angol ABC karaktereire. Ez az egyszerűsítés engedte meg azt, hogy sima szöveggént kezeljek egy adott nyelvi szabályt.

Amennyiben ez nem lett volna, akkor külön indexelt rendszerben kellett volna a szimbólumokat tárolni és akár minden egyes szimbólumhoz egy 2 karakteres megjelenést vagy egyedi képet generálni, hogy egymástól egyértelműen meg lehessen különböztetni a különböző szimbólumok megjelenési képét. Erre azonban nem volt szükség.

A megvalósításban az angol ABC kisbetűi reprezentálják a T szimbólumokat a nagybetűk pedig a NT szimbólumokat (mint megjelenítésben, mint a háttérfolyamatokban). Mivel egy későbbi lehetséges továbbfejlesztést nem akartam figyelmen kívül hagyni, ezért a szabály bal és jobb oldalát elválasztottam egy karakterrel („=”). A jelenlegi rendszer megszorítása – miszerint bal oldalon csak egy darab NT szimbólum lehet – engedte volna azt is, hogy ne legyen a szabály háttér reprezentációjában elválasztó karakter, ezt figyelmen kívül hagytam az előzőek miatt.

Az üres szó szabálynak felvettem a „0” (nulla) karaktert, ez reprezentálja a λ -t.

Az előzőekben leírtak szerint egy szabály rendszerbeli leírása a következőképpen alakul:

$A \rightarrow aA$ szabály leírása „A=aA”

$A \rightarrow \lambda$ szabály leírása „A=0”

A szabályhalmaz egy 1 dimenziós tömb, amely ezeket a „szövegeket” tartalmazza.

Az egyes átalakító konverter függvények a szövegeket dolgozzák fel, és csak a megjelenítéskor konvertálom át a tényleges formázott (a képernyőn is látható) kiírt szövegre. Adatok mentésekor is a háttéradat szerkezete kerül mentésre.

5.3.3. Grafikus felület összeállítása

A szoftver egyik fő moduljának lelkét a grafikus szerkesztő felület alkotja. Több különböző változat után a beépítésre kerül módszert fogadtam el, mint olyan felületet és szerkesztőt, amely megfelel a feladatban kiírt célkitűzésnek. Az egyes változatokban olyan megoldásokat vettem el, amelyeknél problémát okozott az egyes nyíl objektumok megrajzolásánál a nyíl deformációja egyes szögeknél illetve hosszúságoknál. Ilyen gond volt pl. a nyílvég deformációja is.

A végleges megoldásban szétválasztottam a nyíl szárát és a nyíl végét. Ebből a két objektumból rakom össze a tényleges egyben álló nyíl képét. A szárát meg kellett nyújtani és el kellett forgatni, míg a nyílvéget elég volt csak megfelelő irányba forgatni és pozicionálni. Ebből látható, hogy nem megrajzolom a nyilat, hanem grafikus objektumként felhelyezek általános objektumokat, és azokat megfelelően transzformálom a megjelenéshez.

A mellékletben található képek gyűjteményében láthatók azok a különböző színű kis képek, amelyek az alapját adják a felületen megjelenő objektumoknak.

Az állapot objektumok és elfogadó állapot objektumok egyszerűek, hiszen csak egy vagy két kört tartalmaznak. Amikor felkerülnek a felületre, csak megfelelő méretre állítom ezeket és ezzel kész is a megjelenítésük. A hozzájuk tartozó feliratot Z tengelyen eltolva íratom ki az objektumok fölé.

Az átmenet objektumok megrajzolása már jóval bonyolultabb. Nézzük először a nyíl szárát. A szár alapobjektuma egy 2x2 pixeles kis négyzet kép. Ez kellően könnyen nyújtható és forgatható. Tetszőleges méretre nyújtva megőrzi a színét, tehát létrehozható vele egy vonal objektum, amely nyújtás hatására sem torzul. Elforgatva megőrzi a szélességét és nem torzul észrevehetően. A gondot az jelentette, hogy a felületen való forgatás és a pozicionálás nem egy sima feladat. Forgatás hatására az objektum a középpontja körül forog, míg a rajzoláskor egy olyan objektum kellene, amely pl. a bal alsó sarka körül fordul el. Ezt a vizuális helyzetet egy kis matematikával kellett feloldani.

Először megállapítom a tényleges helyet ahová be kell tenni a nyilat, majd onnan visszaszámítom az adatokat arra, ahogyan meg kell adni a forgatást CSS3-ban definiálva. Ezután (mivel még mindig az eredeti objektum bal felső sarka a pozicionálás pontja) kiszámítom az eltolási vektort, amellyel a tényleges helyére került az így már elfogatott objektum.

A számításokat az is nehezítette, hogy az átmenet objektum amelyet megrajzolok nem a hozzá tartozó állapot objektum középpontjából indul ki, hanem annak szélétől távtartással,

ezt is bele kellett építeni a számítás menetébe. Amikor mindez elkészült, akkor még azt is meg kellett oldani, hogy az egymással szembe menő nyilakat úgy jelenítsem meg, hogy azokat ne egymásra, hanem kicsit eltolva egymástól rajzoljam ki.

A előbb említett problémák megoldása után alakult ki egy szép, jól elrendezett megjelenés.

A *funcCreaNyilObject()* függvény végzi egy átmenet objektum igazítását a felületre. A számítás menetét ebben a rutinban lehet részletesen követni. A rutinnak számos paraméterre van szüksége ezekhez a számításokhoz:

1. Ahonnan a nyíl kiindul (x,y koordináta; állapot objektum közepe)
2. Ahová a nyíl mutat (x,y koordináta; állapot objektum közepe)
3. A kiinduló helyzettől való indulási távolság (pixelben, mivel kör alakú az állapot objektum így elég egy értéket megadni)
4. A nyíl végpontjától mért távtartás (pixelben)
5. Melyik objektum kerül megjelenítésre
6. A kiírandó T karakterek (vagy egyéb szöveg)
7. A középtől való eltolás értéke (ha van szembe nyíl, akkor van értéke, ha nincs, akkor 0 értéket vesz fel)

Felmerülhet a kérdés, hogy miért van szükség kiinduló és végállapot közti távtartás külön értékére, amikor ugyanolyan az állapot objektum alakja? Azért, mert amikor éppen rajzolás alatt van egy átmenet objektum, akkor a végállapot az eger aktuális pozíciója (amelyet éppen mozgatunk), és az csak egy pont, amelyen nincs egy állapot objektum, tehát ekkor a záró távtartás értéke 0 pixel.

Az átmenet objektumnak van felirata is. A felirat az átmenet objektumhoz tartozó T szimbólumok listájából áll. Ha több darab szimbólum tartozik egy átmenet objektumhoz, akkor a kiírásakor vesszővel elválasztva jelenítem meg azokat. A megjelenítés a következő szabályokat követi:

1. Mindig a nyíl irányának megfelelően jobb oldalra kerül kiírásra az információ.
2. A szöveg az átmenet objektum középpontjához van igazítva hosszában.
3. Az objektumtól fixen meghatározott távolságban kezdődik vagy záródik a szöveg kiírása.
4. A szöveg kiírása nem fordítódik el a nyíl elforgatásával, mindig vízszintesen történik a megjelenítése.

Ezeket a szabályokat követve a szoftver automatikusan kiszámítja a szöveg megjelenítési helyzetét, így nincs szükség emberi beavatkozásra vagy a szövegek elhelyezésének módosítására manuálisan. Amennyiben a felhasználónak mégis problémát okozna a megjelenített információ helyzet, esetleg takarásba kerülnek információk, akkor az állapot objektumok elmozgatásával tudja korrigálni az automata megjelenési képét. Minden állapot objektum elmozdításakor a hozzá kapcsolódó átmenet objektumok automatikusan újrarajzolódnak és kijavítódik a megjelenésük.

Az objektumok jelölésére (ID-jére) kidolgoztam egy szabályt. Mindegy, hogy állapot vagy átmenet objektumról van szó a rendszer a következő szabad sorszámot (*iAllapotMax* változó) adja új objektum létrehozásakor.

Állapot objektum ID-je: „*allapot[sorszám]*”.

Átmenet objektum esetében több kiegészítés is van mivel ez több részből áll össze:

Ugyanazon állapotra mutató átmenet esetében:

Visszakanyarodó nyíl: „*nyil[sorszám]v*”

Nyíl felirata: „*nyil[sorszám]b*”

Két állapot közötti átmenet esetében:

Nyíl vonala: „*nyil[sorszám]v*”

Nyíl feje: „*nyil[sorszám]n*”

Nyíl felirata: „*nyil[sorszám]b*”

Az éppen rajzolt új átmenet objektum esetében:

Nyíl vonala: „*osszekotv*”

Nyíl feje: „*osszekotn*”

Ezeket az ID-ket használva minden pillanatban pontosan elérhetővé válik az összes rajz objektum a szerkesztő mezőben. Így nem volt szükség arra, hogy a grafikus felületnél ugyanúgy járjak el mint a nyelvi modul esetében, hogy meg van különböztetve a háttér adat és a megjelenített adat. A grafikus felületen a grafikai objektumok az összes szükséges adatot magukban hordozzák, ezt figyelembe véve, ha valamilyen adatra szükségem van egyenest a grafikus objektumoktól kérdezem le azokat. A *funcGetAllapotData()* függvény arra hivatott, hogy egy adott sorszámhoz tartozó állapot objektum összes adatát (beleértve a hozzá kapcsolódó átmenet objektum adatait is) összeszedje. Egy darab állapot objektumhoz a következő adatok tartoznak:

1. Állapot objektum sorszáma (értéke mindig ≥ 1)
2. Bal felső sarok left
3. Bal felső sarok top
4. NT szimbólum
5. Vég állapot-e? (0-Nem; 1-Igen)
6. Kiinduló átmenet objektumok adatai (több is lehet, tömb)

Egy darab átmenet objektum adata:

1. Átmenet objektum sorszáma
2. Állapot 1 objektum sorszáma (kiinduló objektum)
3. Állapot 2 objektum sorszáma (amely objektumra mutat)
4. Forgatási szög (fokban és csak ugyanarra az átmenet objektumra mutató esetén)
5. T szimbólumok (több is lehet, tömb)

5.3.4. Nyelvi szabályokat feldolgozó rutinok

A nyelvi szabályhalmaz átalakító függvények általában tartalmaznak egy inicializáló (indító) függvényt és egy feldolgozó függvényt. Amíg az indító vagy futtató függvény egyszer fut le és az a feladata, hogy beállítsa a számításhoz szükséges alapértékeket és az eredmény kiírja a felhasználó számára, addig a feldolgozó függvény valamilyen rekurzív folyamatot végez, és a szabályhalmaz minden egyes szabályát akár többször is felhasználja a konverziós folyamatban. A rekurzivitás azért fontos, mert egy szabályban ugyanazon NT szimbólum többször is előfordulhat, és akkor minden egyes NT szimbólumon el kell végezni ugyanazt az ellenőrző vagy helyettesítő műveletet.

A számos függvény közül csak egyet említek most részletesebben a szavak előállításának rutinját. A indító függvénye a *funcSzabalyFeldolgoz()* függvény. A feldolgozó függvénye pedig a *funcEgySzabalyMegoldas()* függvény.

Az indító függvény először beállítja a futtatáshoz szükséges korlátozó paramétereket, amelyeket a „Szabályhalmaz feldolgozási műveletek beállításai” blokk mezőiből olvas ki.

```
var MaxMelyseg=SzamMinMax(parseInt($("#maxiteracio").val()),1,50); /* Mennyi mélységig
állítsa össze a szavakat */
var MaxSzo=SzamMinMax(parseInt($("#maxszo").val()),100,10000); /* Mennyi a maximális szavak
száma */
var MaxTime=SzamMinMax(parseInt($("#maxtime").val()),1,86400); /* Maximális futási idő sec-
ben */
var dMax=new Date(new Date().getTime()+MaxTime*1000);
```

Ezután elindítja a feldolgozó függvényt az összes olyan szabály jobb oldalával, amely az *S* szimbólumhoz van rendelve.

```
for(i=0;i<aSzabaly.length;i++){
    if(aSzabaly[i].charAt(0)=="S"){
        av=funcEgySzabalyMegoldas({dmax:dMax,max:MaxSzo,szint:MaxMelyseg,szo:aSzabaly[i].repl
ace(regExpUresSzo,"").substr(2),megoldas:aMegoldas,aLepes:"["+aSzabaly[i]+""]});
        if(!av.error){
            aMegoldas=av.megoldas.clone();
            iEgesz=Math.min(iEgesz,av.egesz);
        }
    }
}
```

Az összes lehetséges lefutás után feldolgozza a kapott eredmény halmazt (*aMegoldas*). Ekkor történik az egyedi szavak előállítása (ha szükséges) és ezen belül a egyes egyedi szavak darabszámának megszámlálása is. Végezetül az eredmények megjelenítése a *#valasz* blokkba íródik ki.

A feldolgozó függvény arra hivatott, hogy megkeresse a lehetséges szabályokat, amelyekkel az aktuális szabály NT szimbólumait helyettesíteni tudja. Ehhez minden lefutáskor a teljes szabályhalmazt átvizsgálja. Lényeges, hogy ebben a fázisban az üres szó szabály jelölése eltávolításra kerül, mert a keletkezett szóban ilyen szimbólum nem szerepelhet.

```

for(i=0; (i<aSzabaly.length) && (aMegoldas.length<param.max); i++) {
    sz=aSzabaly[i].replace(regExpUresSzo, "").trim();
    c=sz.charAt(0);
    j=sSzo.indexOf(c);
    if(j!=-1) {
        /* Összeállítjuk a továbbhaladás naplózását */
        a=aLepes+" [" +sSzo.substr(0,j)+"<b>" +sSzo.substr(j,1)+"</b>" +sSzo.substr(j+1)+"",
            "+aSzabaly[i]+""]";

        av=funcEgySzabalyMegoldas({dmax:dMax,max:param.max,szint:iSzint-1,szo:new
            String(sSzo.substr(0,j).trim()+sz.substr(2).trim()+sSzo.substr(j+1).trim()),
            megoldas:aMegoldas,aLepes:a});
        if(!av.error) {
            aMegoldas=av.megoldas.clone();
            iEgesz=Math.min(iEgesz, av.egesz);
        }
        b=true;
    }
}
if((b==false) && (aMegoldas.length<param.max)) {
    if(sSzo.length==0) sSzo=sUresSzoStr;
    /* Az aktuális kész napló sor lezárása a keletkezett szóval */
    aMegoldas.push(new Array(sSzo.replace(/ /g, "").trim(), aLepes+" [" +sSzo+""]));
}

```

Ezt egészen addig végzi, ameddig el nem fogynak az NT szimbólumok vagy valamely korlátozó feltétel meg nem állítja a rutin futását. Az előállított szavak halmazában nem szerepelnek azok a szavak, amelyek előállítását valamely korlátozó feltétel megszakította, csak azok amelyeknek feldolgozása teljes egészében lezáródott (nincs benne NT szimbólum).

A feldolgozó futása során összeállítja azt az információt is, amellyel megmutatja a felhasználó számára az adott szó előállítási folyamatát.

5.3.5. JavaScript kiterjesztése

A JavaScript nyelv lehetőséget kínál arra, hogy az alap változó objektumokat kiegészítse a fejlesztő sokszor használt egyéb funkciókkal (úgynevezett prototípus deklarációval). Számos kiegészítést használok a rendszerben, amelyek a JavaScript függvénykönyvtár „kiegészítések” blokkjában vannak definiálva („index1.js” fájl).

Ezek az általában nagyon rövid (általában pár soros) kis programrészletek több változatban is fellelhetők az interneten, különböző apró célok megvalósítására hozzák létre ezeket és szabadon felhasználhatóak. Én már rég óta használok ilyen kiegészítéseket a fejlesztéseim során és számos függvényt én magam készítettem. A következőkben ismertetett kiegészítések példák ezekre a függvényekre.

A *String* objektumot kiegészítő „delChar()” függvény, amely arra hivatott, hogy egy megjelölt karaktert eltávolítson egy szövegből. Erre azért volt szükség, mivel az egyes nyelvi szabályok kezelése szöveggént jelenik meg a szoftverben, így ha egy szabályból el akarok távolítani egy NT szimbólumot, akkor ezt a kiegészítést használok. Pl.:

```
sz=sSzabaly1.delChar(j);
```

Az *Array* objektumra létrehozott „clone()” kiegészítés sokszor használt. Azt biztosítja, hogy egy több dimenziós tömbről is tudok másolatot készíteni biztonsággal egy egyszerű utasítás használatával. Például a szabályhalmaz másolásakor sokszor használom. Pl.:

```
var aNoTerm=av.aNoTerm.clone();
```

A *Date* objektumra formázási kiegészítés van létrehozva („format()”), amellyel megfelelő formátumban tudok kiírni egy dátum-idő értéket. A függvény tetszőleges formátummal paraméterezhető az általánosságban használt dátum formázó karakterekkel (y – év; M – hónap; d – nap; h – óra; m – perc; stb.). Pl.:

```
localStorage.setItem("shd"+iSorsz,d.format("yyyy.MM.dd hh:mm:ss"));
```

A *Math* objektumra a grafikus objektumok elforgatásához szükséges radián és fok értékek közötti konverterek vannak beépítve („degrees()”, „radians()”). Erre az egyéb szögfüggvényekben (pl.: *sin()*) megadható radián értékű paraméterek miatt volt szükség. Az itt bemutatott két függvény akár külön deklarációban (egyedi függvényekként) is szerepelhettek volna, mégis jogosnak éreztem, hogy a *Math* részeként legyenek deklarálva a többi szögfüggvény „oldalán”.

5.3.6. Egyéb

Ebben a blokkban megemlítek olyan megoldásokat a szoftverből, amelyek érdekes részeket tartalmaznak.

5.3.6.1. Kiválasztható T és a NT szimbólumok kezelése

A T és a NT szimbólumok előre kiválaszthatóak a nyelvi modulban. Erre az ABC betűit tartalmazó gombok szolgálnak. Mivel ez összesen 46 gombot jelent, ezért ezen gombok létrehozására PHP kódot használtam.

```
<div id="trterm"><?php for($i=1;$i<=26;$i++)if(($i!=9)&&($i!=12)&&($i!=15))echo "<button  
id=\"term{$i}\">".chr($i+96)."</button>";?></div>  
<div id="trnoterm"><?php for($i=1;$i<=26;$i++)  
if($i==19)echo "<input type=\"button\" id=\"noterm{$i}\" value=\"\".chr($i+64).\"\"  
class=\"select\" />";  
elseif(($i!=9)&&($i!=12)&&($i!=15))echo "<button id=\"noterm{$i}\">".chr($i+64)."</button>\"  
;?></div>
```

Az *S* szimbólum törlésének megakadályozására bevezettem az *input type="button"* típust. Látható, hogy a gomb önmagában nem tartalmaz kezelő függvényt, erre JQuery használók.

```
$("#trterm button").click( function(){  
    funcTNTclick({id:$ (this).attr("id")});  
});  
$("#trnoterm button").click( function(){  
    funcTNTclick({id:$ (this).attr("id")});  
});
```

Mindkettő típus gombjainak lenyomását ugyanaz a *funcTNTclick()* függvény végzi el. Ez a rutin több különböző szükséges feladatot hajt végre. Először is beállítja vagy eltörli az adott gombon a *select* stílust.

```
$("#"+sID).toggleClass("select");
```

Ez a stílus gondoskodik a piros színnel való megjelenésről, és ez hordozza azt az információt is, hogy ki van jelölve az adott szimbólum. A függvény a továbbiakban frissíti az új szabály készítő választó listákat (*funcNewKeszitoFresh()* függvény meghívása), és ha törlésre került a szimbólum akkor eltávolítja a már nem szükséges szabályokat.

Megjegyzés: A PHP kódokat ki lehetett volna váltani JavaScript kóddal is, ezáltal elvileg statikus oldalként is használható lenne az oldal. Ez nem volt cél, így ezzel nem foglalkoztam, de egy esetleges továbbfejlesztés alkalmával – jelenleg nem látom akadályát – ezt meg lehet valósítani.

5.3.6.2. Szabályok szintaktikai/logikai formátumának ellenőrzése

Ez azért került ide, mert elvileg nem is lenne rá szükség, de bent maradt a szoftverben (*funcSzabalyEllenoriz()* függvény). A kezdeti fejlesztés során volt szükség arra, hogy ellenőrizzem, hogy nem tartalmaz-e valamilyen olyan szabályt a szabályhalmaz, amely nem felel meg az előírásoknak. Később, amikor kialakult a tényleges szabály szerkesztő felület az összes korlátozással és automatikus szimbólumfeltöltéssel, akkor a szabályban korábban létrehozott hibák egyszerűen eltűntek. Ettől függetlenül nem vettem ki a kódból ezt a függvényt, azért mert egy esetleges mentés / visszatöltés adatsérülés esetén ez a függvény megakadályozza a szoftver hibás működését.

A rutin átvizsgálja a teljes szabályhalmazt, és a hibás (részleges) szabályokat kijavítja, vagy az olyan szabályokat, amelyek nem felelnek meg a 2-es vagy 3-as kategóriának, azokról hibaüzenetben tájékoztat. Vizsgált hibák:

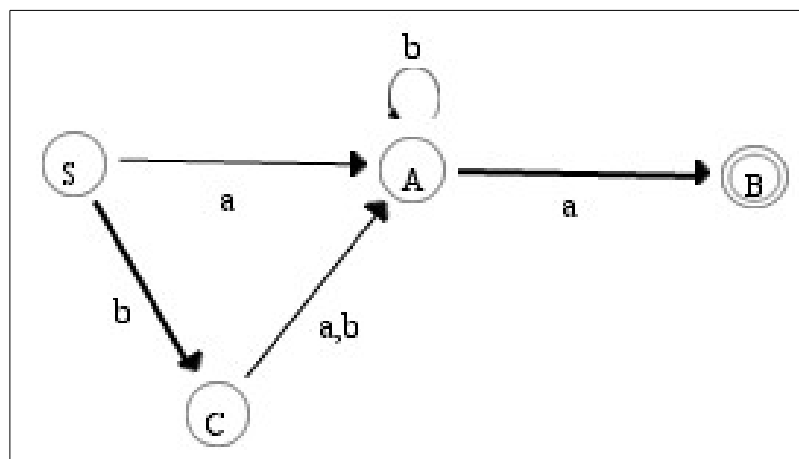
1. Idegen szimbólum megléte (Pl.: $\$ \rightarrow aA, A \rightarrow aEb$)
2. Hiányos szabály (pl.: $A \rightarrow$)
3. NT hiány (pl.: $\rightarrow aA$)
4. T szabály (pl.: $a \rightarrow bC$)
5. „Fordított” üres szó szabály (pl.: $\lambda \rightarrow A$)
6. Kategória hiba, 0-s vagy 1-es szabály (Pl.: $AB \rightarrow aB$)
7. Egyszerűen teljesen üres szabály (amikor nem egyetlen karaktert sem)

5.4. Szoftver tesztelése

5.4.1. Helyes működés tesztelés folyamata

Kialakítottam egy tesztelési tervet, amely alkalmazásával minden lényeges funkció helyes működése felmérhető. A következőben ezeket a lépéseket ismertetem.

1. A „Végtes automata modul” menüpont kiválasztása.
Eredmény: Megjelenik a modul felülete.
2. Húzzon le 4 db „Új állapot” (kör) objektumot a szerkesztő felületre.
Eredmény: minden lehúzáskor ott marad a felületen egy állapot objektum amely más-más azonosítót kap (S, A, B, C)
3. A szerkesztő felületen egy állapot objektumra (körre) tegye rá az egeret.
Eredmény: Meg kell jelenjen egy kék pötty. A pöttyből nyilat lehet húzni egy másik állapot objektumhoz.
4. Kösse össze a következőket: $S \rightarrow A$; $A \rightarrow B$; $S \rightarrow C$; $C \rightarrow A$; $A \rightarrow A$ (Önmagára mutató kapcsolat).
Eredmény: Húzáskor megjelenik a nyíl, és ahogyan mozgatjuk az egeret szépen követi azt a kiinduló állapot objektum körül. Ha egy másik állapot objektumon engedjük el, akkor létrehoz a két objektum közé egy kapcsolat nyilat (átmenet objektum). Fel is címkézi a nyilat a következő lehetséges szimbólummal.
5. Kattints a B jelű állapot objektumra. Majd a szerkesztő alatt kapcsolja be az „elfogadó állapot”-ot.
Eredmény: Dupla körös állapot objektum megjelenése
6. Kattints a $C \rightarrow A$ átmenet objektumra (nyílra) és szerkesztő alatt adjon hozzá plusz egy átmenetet.
Eredmény: Két betű vesszővel elválasztva jelenik meg az átmenet mellett, pl.: a,b



27. Ábra: Teszt automata kézi változata

Eredmény: Két betű vesszővel elválasztva jelenik meg az átmenet mellett, pl.: a,b

7. Tesztelje le az automata működését, indítsa el a „Automata működésének bemutatása”-t.

Eredmény: Vizuálisan és szövegesen bemutatja az automata működését

8. Mentse el az automatát valamilyen néven.

Eredmény: Az „Eltárolt automaták” között megjelenik a mentett anyag.

9. Konvertálja át nyelvtanná az automatát: „Automata konvertálása nyelvtanná”.

Eredmény: Átvált a rendszer a „Reguláris nyelv modul”-ra és ott megjelenik a nyelvtan.

$G = (\{A, B, C, S\}, \{a, b\}, \{A \rightarrow aB, A \rightarrow bA, B \rightarrow \lambda, C \rightarrow aA, C \rightarrow bA, S \rightarrow aA, S \rightarrow bC\}, S)$

10. A nyelvtan fülön található anyagot konvertálja vissza automatává („Szabály konvertálása automatává” kiválasztása és „Feldolgozás indítása” gomb aktiválása).

Eredmény: „Az automata elkészült, lépjen át az automata modulhoz!” szöveg megjelenik.

11. Nézze meg az automata képét.

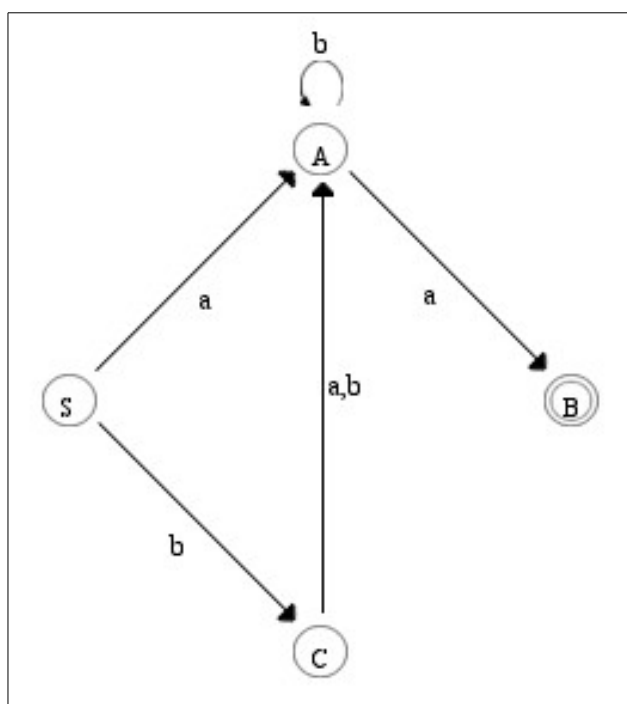
Eredmény: Az átalakított automata megjelenik (szerkezete ugyanaz mint amit korábban szerkesztettünk, csak más elrendezésben van!)

12. Betöltés tesztelés. Töltse vissza az elmentett anyagot.

Eredmény: Visszaáll az eredeti elrendezés.

13. Nyelv váltás tesztelése. Kapcsoljon át másik nyelvre és vissza.

Eredmény: A szövegek átcserélődnek a kiválasztott nyelvi anyagra.



28. Ábra: Teszt automata konvertált változata

5.4.2. Működés helyesség vizsgálatának eredménye

Számos operációs rendszeren és böngészőben lett tesztelve a rendszer, így elég általános képet kaptam arról, hogy mennyire használható a program.

A mellékletben található táblázatban összefoglaltam a különböző böngészőkben végzett működés vizsgálat eredményét (alább).

Alapvetően elmondható, hogy jó pár széles körben használt webböngészőben és operációs rendszeren (bele értve mobil eszközt is) teljes funkcionalitással lehet használni a szoftvert. Míg a hibás működések elsősorban a grafikai elemek használatában jelentkeznek, de ezen böngészőkben is a reguláris nyelvi modul teljes funkcionalitással használható.

5.5. Szoftver telepítés

Alapkövetelmény: a rendszer működtetéséhez szükséges egy web kiszolgáló motor, amelyben engedélyezve van a php futtató funkció.

Készítsünk egy web tárhelyet, majd másoljuk be a program állományait a létrehozott futtatási könyvtárba. Ha a web kiszolgáló tudja az „*index.php*” fájl futtatását, akkor készen is vagyunk azonnal használható a rendszer. Ha nem, akkor ezt állítsuk be a web kiszolgáló rendszerben.

IRODALOMJEGYZÉK

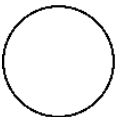
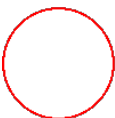
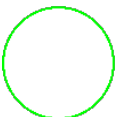
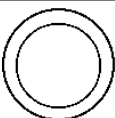
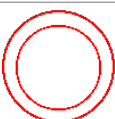
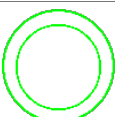
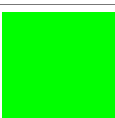
- [1] Ésik Zoltán, Gombás Éva, Iván Szabolcs: Automaták és formális nyelvek példatár
2011, Typotex Kiadó, ISBN 978-963-279-495-2
- [2] Vajk István: Informatika 2; Automaták és nyelvek
2010, Jegyzet
https://wiki.sch.bme.hu/images/b/bd/Info2_Automat%C3%A1k_%C3%A9s_nyelvek_jegyzet.pdf, 2015.10.10
- [3] Bach Iván: Formális nyelvek
2002, Typotex Kiadó, ISBN 9639132 92 6
- [4] Friedl Katalin: Nyelvek és Automaták
2010, Jegyzet
https://wiki.sch.bme.hu/images/2/23/Nyau_jegyzet_2010.pdf, 2015.11.25
- [5] Váraljai Fruzsina: The Butterfly Project
<http://www.varaljai.hu/butterfly/index.html>, 2015.11.25
- [6] Kallós Gábor: A reguláris nyelvek jellemzése 2.
2014, Jegyzet
http://www.sze.hu/~kallos/fny-szamtud/fny-06u-1_or.pdf, 2015.11.25
- [7] Révész György: Bevezetés a formális nyelvek elméletébe
Akadémiai Kiadó, 1979, ISBN: 963-05-1720-5
- [8] Tichler Krisztián: Nyelvtani transzformációk
<http://www.cs.elte.hu/~tichlerk/fny/gy06H.pdf>, 2016.04.20
- [9] Dömösi Pál, Falucskai János, Horváth Géza, Mecsei Zoltán, Nagy Benedek:
Formális Nyelvek és Automaták
2011, Jegyzet
http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_formalis_nyelvek_es_automatak/index.html, 2016.04.20
- [10] Kallós Gábor: Formális rendszerek, Generatív nyelvtanok, Chomsky-féle
nyelvosztályok
2014, Jegyzet
http://www.sze.hu/~kallos/fny-szamtud/fny-02u-1_or.pdf, 2016.04.20
- [11] Fülöp Zoltán: Formális nyelvek és szintaktikus elemzésük
2011, Polygon kiadó, Szeged

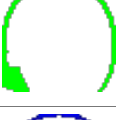
Felhasznált szoftverfejlesztési segédletek:

- [12] jQuery dokumentáció
<http://api.jquery.com/>
- [13] jQueryUI dokumentáció
<http://api.jqueryui.com/>
- [14] W3Schools.com (JavaScript, HTML5, CSS3)
<http://www.w3schools.com/>
- [15] jscompress.com - Online JavaScript kód tömörítő oldal
<http://jscompress.com/>
- [16] CSS Compressor - Online CSS kód tömörítő oldal
<http://csscompressor.com/>
- [17] HTML/PHP Compressor - Online HTML/PHP kód tömörítő oldal
<http://htmlcompressor.com/>

MELLÉKLETEK

A szoftver képi anyagai

Minta	Fájl neve és mérete	Használata
	allapot.png 100px*100px	Állapot objektum alap állapotban
	allapotp.png 100px*100px	Állapot objektum kijelölt állapotban
	allapotj.png 100px*100px	Állapot objektum bemutatókor kijelölt állapotban
	allapotz.png 100px*100px	Elfogadó állapotú állapot objektum alap állapotban
	allapotzp.png 100px*100px	Elfogadó állapotú állapot objektum kijelölt állapotban
	allapotzj.png 100px*100px	Elfogadó állapotú állapot objektum bemutatókor kijelölt állapotban
	vonals.png 2px*2px	Átmenet objektum rajzolásához a nyíl vonala alap állapotban
	vonalsp.png 2px*2px	Átmenet objektum rajzolásához a nyíl vonala kijelölt állapotban
	vonalsj.png 2px*2px	Átmenet objektum rajzolásához a nyíl vonala bemutatókor kijelölt állapotban
	vonalsb.png 2px*2px	Átmenet objektum rajzolásához a nyíl vonala bemutatókor véglegesen kiválasztott állapotban

Minta	Fájl neve és mérete	Használata
	von4.png 8px*5px	Átmenet objektum rajzolásához a nyíl feje alap állapotban
	von4p.png 8px*5px	Átmenet objektum rajzolásához a nyíl feje kijelölt állapotban
	von4j.png 8px*5px	Átmenet objektum rajzolásához a nyíl feje bemutatókor kijelölt állapotban
	von4b.png 8px*5px	Átmenet objektum rajzolásához a nyíl feje bemutatókor véglegesen kiválasztott állapotban
	von5.png 50px*50px	Ugyanazon állapotra mutató átmenet objektum alap állapotban
	von5p.png 50px*50px	Ugyanazon állapotra mutató átmenet objektum kijelölt állapotban
	von5j.png 50px*50px	Ugyanazon állapotra mutató átmenet objektum bemutatókor kijelölt állapotban
	von5b.png 50px*50px	Ugyanazon állapotra mutató átmenet objektum bemutatókor véglegesen kiválasztott állapotban
	nyilk2.png 10px*10px	Állapot objektum rajzolás indításának jelzése ill. állapot objektum forgatásának jelzése
	hu_hu.png 100px*50px	Magyar nyelv jelzéséhez zászló; Forrás: https://commons.wikimedia.org/wiki/File:Flag_of_Hungary.svg
	en_gb.png 100px*50px	Angol nyelv jelzéséhez zászló; Forrás: https://commons.wikimedia.org/wiki/File:Flag_of_the_United_Kingdom.svg
	de_de.png 100px*50px	Német nyelv jelzéséhez zászló; Forrás: https://commons.wikimedia.org/wiki/File:Flag_of_Germany.svg

Minta	Fájl neve és mérete	Használata
	favicon.ico 32px*32px	Az oldal ikonja Készítéséhez „Révész György: Bevezetés a formális nyelvek elméletébe”[7] című könyvének borítója adta az ötletet

Szoftver tesztelés eredménye

Böngésző	Verzió szám	Operációs rendszer	localStorage	Grafikus megjelenés
Google Chrome	44.0.2403.155	Debian Linux (32bit)	Ok	Ok
Google Chrome	48.0.2564.103 m	Win XP	Ok	Ok
Google Chrome	48.0.2564.109 m	Win 7	Ok	Ok
Google Chrome	49.0.2623.105	Android 5.0.2	Ok	Ok
Google Chrome	49.0.2623.112 m	Win 7	Ok	Ok
Google Chrome	49.0.2623.112 m	Debian Linux (64bit)	Ok	Ok
Google Chrome	50.0.2661.75 m	Win 8.1	Ok	Ok
Opera	35.0.2066.37	Win XP	Ok	Ok
Opera	35.0.2066.37	Win 7	Ok	Ok
Safari	5.1.7 (7534.57.2)	Win XP	Ok	Ok
Safari	5.1.7 (7534.57.2)	Win 7	Ok	Ok
HTC böngésző	7.20.2522332440	Android 5.0.2	Ok	Ok
Mozilla Firefox	44.0.1	Win XP	Ok	Nyilak rosszak
Mozilla Firefox	45.0.2	Win 7	Ok	Nyilak rosszak
Mozilla Firefox	45.0.2	Win 8.1	Ok	Nyilak rosszak
Internet Explorer	8.0.6001.18702	Win XP	Ok	Nyilak és körök is rosszak
Internet Explorer	11.0.9600.17959	Win 7	Ok	Nyilak rosszak
Internet Explorer	11.0.9600.18163	Win 7	Ok	Nyilak rosszak
Internet Explorer	11.0.9600.18283	Win 8.1	Ok	Nyilak rosszak
Iceweasel	38.2.0	Debian Linux (32bit)	Ok	Nyilak rosszak
Iceweasel	38.7.1	Debian Linux (64bit)	Ok	Nyilak rosszak