

Tartalom

1	Prolan felhasználói dokumentáció	2
1.1	Munkalapok	2
1.1.1	Táblázat - Nyelvtan.....	2
1.1.2	Táblázat - Programozott nyelvtan	4
1.1.3	Program – Programozott nyelvtan.....	4
1.1.4	Program – Turing-gép, Veremautomata	5
1.2	Beállítások	6
1.2.1	Általános.....	6
1.2.2	Programozott nyelvtan beállításai	7
1.2.3	Turing-gép, Veremautomata.....	7
1.2.4	Könyvtárak	8
1.2.5	Stílusok	8
1.3	A Prolan nyelv	8
1.3.1	Hello World!	8
1.3.2	Fordítási direktívák, kommentek	10
1.3.3	Típusok	11
1.3.4	Deklarációk.....	11
1.3.5	Beépített konstansok	13
1.3.6	Logikai kifejezések.....	13
1.3.7	Utasítások.....	14
1.3.8	Blokkok	15
1.3.9	Makrók	16
1.4	Speciális nyelvtanok, automaták programozása.....	17
1.4.1	Közönséges nyelvtanok	17
1.4.2	Mátrixnyelvtanok	18
1.4.3	Lindenmayer-rendszerek	19
1.4.4	Automaták.....	20
1.4.5	Turing-gép	21
1.4.6	Véges automata	24
1.4.7	Veremautomata	25
2	Irodalomjegyzék.....	27

1 Prolan felhasználói dokumentáció

A program egy 32 bites Windows alkalmazás, mely Delphi 2007 fejlesztői környezetben készült. Windows 7 operációs rendszer és 1280*1024-es képernyőfelbontás ajánlott (a felhasználói felület erre lett optimalizálva), de működik Windows XP rendszeren is. Általában a program memóriaigénye 10-20 MB, bizonyos ritka esetekben azonban felmehet több száz MB-ra is, ilyenkor ajánlott egyszerre csak kevés munkalap használata.

A program külön telepítést nem igényel.

1.1 Munkalapok

A felhasználói felület munkalapokra osztott. Új munkalapot a Fájl/Új menüvel adhatunk hozzá. Az aktuális munkalapot bezárhatjuk Ctrl+F4 billentyűkombinációval, vagy a Fájl/Bezárás menüponttal. A Fájl/Mind bezárása értelemszerűen mindegyik munkalapot bezárja.

A munkalapok alapvetően az alábbi két típusba tartoznak:


Táblázatok: A közönséges és programozott nyelvtan különböző paramétereit (ábécé, szabályok stb.) egy táblázat kitöltésével adhatjuk meg. Ennek a típusnak előnye, hogy használata egyszerű, a formális nyelvtanok ismeretén kívül semmilyen előismeretet nem igényel.

Program: Programozott nyelvtant, Turing-gépet ill. veremautomatát adhatunk meg egy erre szolgáló programnyelv segítségével. Előnye, hogy a nyelv kifejezőereje lehetővé teszi sokkal nagyobb és bonyolultabb nyelvtanok létrehozását, ill. ugyanaz a nyelvtan rövidebben fogalmazható meg programként, mint táblázattal. Könnyen modellezhetünk pl. mátrixnyelvtanokat vagy Lindenmayer rendszert is. Az automatákat (T-gép, VA) csak programként adhatjuk meg, táblázattal nem.

Az alábbi fejezetekben áttekintjük a különböző típusú munkalapokat.

1.1.1 Táblázat - Nyelvtan

Ezen a munkalapon közönséges formális nyelvtanokat adhatunk meg táblázatos formában. Használat:

1. A bal oldali táblázatokban adjuk meg a nyelvtan szimbólumait és szabályait. A táblázat alján megjegyzést fűzhetünk a nyelvtanhoz, pl. a generált nyelv leírását, vagy utasításokat a használatra.
2. Futtassuk le a nyelvtant a  gombra kattintva.
3. A futtatás gyakorlatilag azonnal lezajlik, és az ablak jobb oldalán böngészhetjük ennek eredményét, vagyis a generált

Start szimbólum	Nemterminálisok	Terminálisok
S	A B L R	a
Bal oldal	Jobb oldal	
S	A L a B	
A L	A R	
R B	L B	
a L	L a a	
R a	a a R	
A L	R	
R B	L	
A L	eps	
R B	eps	

Generált nyelv: $a^*(2^n)$
Az A L -> eps szabályt csak akkor szabad használni, ha már nincs B.
Hasonlóan R B -> L csak akkor, ha már nem szerepel A.

1. ábra: Közönséges nyelvtan megadása táblázattal

levezetéseket.

A szimbólumok megadására vonatkozó szabályok:

- A szimbólum az angol ábécé betűiből és számjegyekből állhat. Kezdődhet számjeggyel, és állhat csak számjegyből is.
- Lista esetén (pl. terminális jelek felsorolása) a szimbólumokat szóközzel kell elválasztani.
- A görög ϵ használható rövidítése: **eps**.

1.1.1.1 Lista nézet

A lista nézet célja egyetlen levezetés megjelenítése. Az elemeket automatikusan generálja a program bizonyos szabályok szerint (ld. Beállítások 1.2), de az elemekre kattintva módosíthatjuk is a levezetés menetét. A listának két oszlopa van:

Mondatforma: Az adott lépéshez tartozó mondatforma. Abban az esetben, ha a megelőző lépés során a szabály több pozíción is végrehajtható volt (pl. az AAB mondatformára az $A \rightarrow a$ szabály két helyen is illeszthető), akkor a mondatformára kattintva megválaszthatjuk a kívánt pozíciót.

Szabály: Ez az oszlop tartalmazza a nyelvtani szabályt, amit

alkalmazunk az aktuális mondatformára. Amennyiben több szabály is alkalmazható, az elemre kattintva megválaszthatjuk a nekünk megfelelőt.



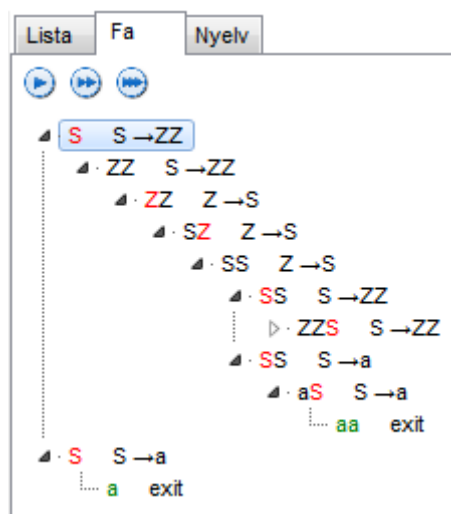
2. ábra: Lista nézet

A lista nézet tetején található egy egyszerű animációs rész, amivel lejátszhatjuk a levezetést lépésről lépésre. A lejátszást a szokásos gombokkal irányíthatjuk.

1.1.1.2 Fa nézet

A fa nézet célja az összes lehetséges levezetés megjelenítése kibontható fa struktúrában. A nyelvtan lefuttatása után a fa egyedül a kezdő konfigurációt tartalmazza, melyet ha a mellette lévő jellel „kibontunk”, megjelennek a belőle közvetlenül levezethető konfigurációk.

A panel tetején lévő gombokkal 4, 16 vagy 64 lépés mélységben, rekurzív módon bonthatjuk ki a kijelölt részét.



3. ábra: Fa nézet

1.1.1.3 Színkódok

A program piros színnel jelöli azokat a részzavakat, melyekre az aktuális szabály illesztése történt. Ha Fa nézetben egyszerre több illesztés is tartozik egy elemhez, ekkor ezek közül az első számít. Ha egy szó csupa terminális jelből áll, vagyis a generált nyelv részét képezi, akkor a színe zöld (ld. 3. ábra **aa** és **a** szavai). Ha egy szóra nem folytatható a levezetés (ezek „sikertelen” levezetés-ágaknak tekinthetők), akkor a színe piros (ld. **Hiba! A hivatkozási forrás nem található.** utolsó eleme: **aaaaB**).

1.1.1.4 Nyelv nézet

A nyelv nézet célja a nyelvtan által generált nyelv néhány elemének megjelenítése. Mivel ez a számítás időigényes lehet (a hatékonyság nagyban függ a beállításoktól (1.2)), csak szükség esetén indul el: akkor, ha a felhasználó a Nyelv fülre kattint. Egy idő után megjelenik egy felugró ablak mely az eltelt időt jelzi, valamint a Cancel-gomb megnyomásával lehetőséget ad a számítás leállítására. A generált elemek ekkor sem vesznek el, csak nem keletkeznek továbbiak. A számítás befejeztével az eredményeket hosszúságuk alapján rendezi sorba, másodlagos rendezési szempont szerint pedig lexikografikusan.

1.1.2 Táblázat - Programozott nyelvtan

Ezen a munkalapon programozott nyelvtanokat modellezhetünk. Használata megegyezik a közönséges nyelvtanokéval, azzal a különbséggel, hogy most meg kell adnunk minden szabályhoz egy címkét is (ID oszlop), valamint a σ , φ halmazokat. Ezen halmazok jelentése: ha a levezetés egy lépésében pl. az alábbi ábrán a 3. címkéjű szabályt hajtottuk végre, és a végrehajtás sikeres volt (vagyis a mondatformában szerepelt B szimbólum, amit lecserélhettünk Bb-re), akkor a következő lépésben a 4 vagy a 2 címkéjű szabállyal kell folytatni a levezetést. Ha a végrehajtás sikertelen volt (nem szerepelt B a mondatformában), akkor a φ oszlopból kell címkét választanunk. Ez az oszlop a példában most egyedül az **exit**-et tartalmazza, ami azt jelenti, hogy ekkor a levezetés véget ér.


Start szimbólum		Nemterminálisok		Terminálisok
S		A B		a b
ID	Bal oldal	Jobb oldal	σ	φ
1	S	A B	2 4	exit
2	A	A a	3	exit
3	B	B b	4 2	exit
4	A	eps	5	exit
5	B	eps	exit	exit

Generált nyelv: $a^n b^n$, ahol $n \geq 0$

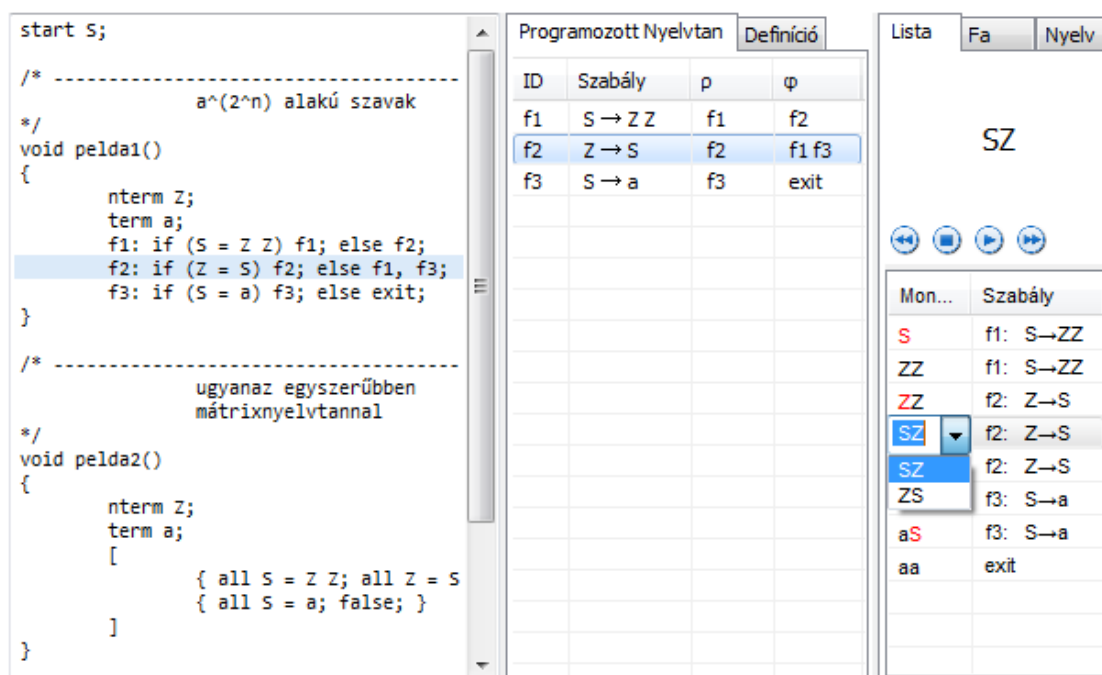
4. ábra: Programozott nyelvtan megadása táblázattal

1.1.3 Program – Programozott nyelvtan

Ezen a munkalapon programozott nyelvtanokat adhatunk meg egy erre szolgáló programnyelv segítségével. Használat:

1. A bal oldali kód-szerkesztő ablakrészben írjuk meg a nyelvtan megadására szolgáló programkódot. A nyelv részletes leírását ld. a 1.3 fejezetben. Üres munkalap létrehozásakor egy előre megírt vázat kapunk, amely a PNY-ok használatához van igazítva.
2. Futtassuk le a programot a  gombra kattintva. A futtatás két lépésben történik:
 - 2.1. A munkalap középső részén megjelennek a kész nyelvtan szabályai. Ha a programkód hibás, hibaüzenetet kapunk, és a nyelvtan nem jön létre. Szintén a középső részen találunk egy „Definíció” fület, amelyre kattintva megtekinthetjük a PNY matematikai definíciónak megfelelő leírását (ennek jelentősége leginkább majd a T-gép és a VA esetén lesz).
 - 2.2. Az előállított nyelvtan generálja a levezetések és a nyelvet, melyeket a jobb oldali ablakrészben böngészhetünk a táblázatos munkalapokéval megegyező módon.

Ha középső ablakrészben a táblázat egy sorára kattintunk (vagyis egy nyelvtani szabályra), az ablak bal oldalán a forráskódban kijelölésre kerül az a programsor, mely az adott szabály leírását tartalmazza. Ha a jobb oldalon a levezetés böngészőben egy konfigurációra kattintunk, az imént említett két ablakrészben szintén kijelölésre kerül a konfigurációhoz tartozó megfelelő sor.



The screenshot displays the macro editor interface with three main panels:

- Left Panel (Code Editor):** Contains two C-like functions. The first, `pelda1()`, uses if-statements to check grammar rules. The second, `pelda2()`, uses a matrix-based approach. The line `f2: if (Z = S) f2; else f1, f3;` in `pelda1()` is highlighted.
- Middle Panel (Grammar Rules Table):** Titled "Programozott Nyelvtan" and "Definíció". It contains a table with 4 columns: ID, Szabály, p, and φ.

ID	Szabály	p	φ
f1	$S \rightarrow ZZ$	f1	f2
f2	$Z \rightarrow S$	f2	f1 f3
f3	$S \rightarrow a$	f3	exit
- Right Panel (Derivations Table):** Titled "Lista", "Fa", and "Nyelv". It shows a table with 2 columns: Mon... and Szabály.

Mon...	Szabály
S	f1: $S \rightarrow ZZ$
ZZ	f1: $S \rightarrow ZZ$
ZZ	f2: $Z \rightarrow S$
SZ	f2: $Z \rightarrow S$
SZ	f2: $Z \rightarrow S$
ZS	f3: $S \rightarrow a$
aS	f3: $S \rightarrow a$
aa	exit

5. ábra: A makrószerkesztő munkalap.

1.1.4 Program – Turing-gép, Veremautomata

Turing-gépek programozása és futtatása megegyezik a programozott nyelvtanokéval.

Különbség a levezetések megjelenítésében van:

- A középső ablakrészben a T-gép állapotai és állapotátmenetei jelennek meg. Minden T-gépnek van egy elutasító és egy elfogadó állapota (melyekre **exit**-tel ill. **accept**-tel hivatkozhatunk), ezeknek nincsenek átmenetei.
- A lista nézetben a mondatforma helyén a szalag állapota látható, rajta pirossal megjelölve a fej pozíciója. Ez nem módosítható, mivel T-gép esetén itt nincsenek

alternatív választási lehetőségek. A szabályra kattintva viszont választhatunk a konfigurációra alkalmazható állapotátmenetek közül.

- A fa nézetben is a szalag állapota látható.
- A nyelv nézetben a T-gép által felismert nyelv (tehát nem az eldöntött nyelv!) néhány eleme látható.
- A levezetés-böngészőben egy elemre kattintva a program kijelöl egy sort a középső ablakrészben, ami egy állapotátmenetet ír le: az aktuális konfigurációból a következő konfigurációba vezető állapotátmenet. Egy átmenetet egyszerre több utasítás is leírhat a forráskódban (olvasás, írás, léptetés), ezek közül mindig az utolsó számít, vagyis ezt jelöli ki a program. A Fa nézetben egy konfigurációnak több gyerek-konfigurációja is van, vagyis egyszerre több átmenetet is ki kellene jelölni, a program ezek közül csak az elsőt jelöli ki.

Különbségek **veremautomaták** esetén:

- A középső ablakrészben láthatóak a veremautomata állapotai és átmenetei. Ezúttal is hivatkozhatunk az **exit** és **accept** állapotokra, most azonban ezek virtuális állapotok, nem tartoznak hozzá az automata matematikai definíciójához, csak a használat megkönnyítését (és egységességét) szolgálják. Ezzel függ össze, hogy az **accept**-re mutató átmenetek verembe és veremki mezője üres, vagyis ezek nem valódi átmenetek.
- Lista és Fa nézetben a szalag helyett az automata bemenete (pirossal megjelölve az aktuálisan olvasott szimbólum) és verme látható, melyek nem módosíthatóak, csak az alkalmazható átmenetek.
- A nyelv nézetben az automata által felismert nyelv néhány eleme látható.

A T-gépek és veremautomaták programozásáról bővebben ld. a 1.3 fejezetben.

1.2 Beállítások

1.2.1 Általános

Determinisztikus lépések elrejtése: Ha ez az opció be van kapcsolva, akkor lista nézetben csak az első és az utolsó állapot jelenik meg, valamint azok, melyekben a levezetés elágazáshoz érkezik, vagyis több irányban is folytatható. Ez az opció hasznos lehet olyan esetekben, amikor áttekinthetetlenül sok lépésből csak azokat akarjuk meghagyni, melyek interakciót tesznek lehetővé.

Animáció, egy lépés időtartama: Megadhatjuk, hogy a lista nézetben lévő animáció egy lépése hány ms alatt játszódjon le.

Mondatforma, verem, szalag maximális hossza: Ha egy levezetésben a mondatforma, verem vagy szalag hossza eléri ezt a küszöböt, a levetés megáll és nem folytatható. T-gép esetén a szalag hossza természetesen (a végtelen) szalagon lévő használt (nem _ jelet tartalmazó) rész hosszát jelenti.

Generálás maximális ideje: ms-ban megadhatjuk mennyi ideig tarthat a megoldások keresése.

Találatok: A nyelv nézetben a találatok maximális száma. Ezt elérve a program nem keres több megoldást.

Lista nézet, lépésszám: Egy futtatás alkalmával hány lépést tegyen meg a lista nézetben látható levezetés (ez nem feltétlenül egyezik meg a megjelenített elemek számával, annál több is lehet). Ha a levezetés még folytatható, a lista alján megjelenik egy Tovább gomb.

Fa nézet, elemszám: Ha a Fa nézet elemszáma elérte ezt a küszöböt, nem folytatódik a rekurzív kibontás (viszont manuálisan, egyesével továbbra is lehet folytatni).

1.2.2 Programozott nyelvtan beállításai

A különböző nézetekben (Lista, Fa, Nyelv) megadhatjuk, hogy a program milyen szabályok szerint generálja ill. szűrje a levezetéseket. A következő lehetőségek adottak:

Pozíció választás nyelvtanok esetén: Egy nyelvtani szabály alkalmazásánál dönteni kell arról, hogy a szabály bal oldalát - amennyiben többször is előfordul az aktuális mondatformában - melyik pozícióra illesszük. Ettől a választástól függően lehet a levezetés legbal, legjobb, véletlen valamint teljes (minden lehetőségre kiterjedő).

Szabály választás: Amennyiben valamely konfigurációnak több gyereke is van, a gép működése nem determinisztikus. Bizonyos esetekben teljesítmény okokból szükség lehet a determinisztikus működés kikényszerítésére. A pozíció választáshoz hasonlóan itt is dönthetünk arról, hogy az első, az utolsó, véletlenül választott szabály kerüljön alkalmazásra, vagy pedig mindegyik. Mivel a kezdő felhasználó számára ez az opció zavaró lehet, ill. ritkán van rá szükség, a felhasználói felületből nem érhető el, csak fordítási direktívával (ld. 1.3.2, **rule** direktíva).

Zsákutcák elrejtése: Ha programozott nyelvtanokkal közönséges nyelvtanokat akarunk modellezni, gyakran szembesülünk azzal a problémával, hogy néhány lépés után a program terminál, mivel olyan szabályt próbál végrehajtani ami nem végrehajtható, miközben más szabályok még végrehajthatóak volnának (a definíciók alapján ilyenkor a közönséges nyelvtan „megkeresi” az alkalmazható szabályt, a PNY viszont erre nem képes). A levezethető szavak halmazán ez nem változtat, de némely levezetések használhatóságát erősen lerontja. Ezt elkerülhetjük azzal, ha elrejtjük a zsákutcákat, vagyis azokat a konfigurációkat, melyek egy lépésben biztosan az **exit** állapotba vezetnek (pontosabban: csak akkor választ a program ilyen utat, ha nincs más lehetőség). Az elrejtés bekapcsolása ugyanakkor zavaró is lehet, mivel nem a definíciónak megfelelő működést eredményez.

1.2.3 Turing-gép, Veremautomata

Bemenetek száma: Turing-gép és veremautomata esetén a program különböző bemenetekre futtatja le az automatát, és vizsgálja hogy terminál-e. Ezek maximális számát adhatjuk itt meg.

Konfiguráció/Bemenet: Ha egy adott bemenetre az automata nem terminál, akkor egy bizonyos mennyiségű konfiguráció után abba kell hagynia a vizsgálatot, és a következő bemenetre kell térnie. Ennek a küszöbnek az értékét adhatjuk meg itt. Természetesen előfordulhat, hogy egy nagyobb küszöb beállítása esetén ugyanaz a bemenet már terminálni fog.

1.2.4 Könyvtárak

Itt megadhatjuk, mely könyvtárakban keresse a fordítóprogram az **#include** utasításban szereplő fájlokat (ha nem használunk **#include**-ot, ez a beállítás lényegtelen). Alapértelmezés szerint a lib alkönyvtár szerepel a listában.

1.2.5 Stílusok

Ezen a panelen beállíthatjuk a programban használt betűtípusokat. Mivel nem minden karakterkészlet tartalmazza a szükséges karaktereket (\rightarrow , ϵ), nem megfelelő betűtípus beállítása esetén hibás lehet a megjelenítés. Külön adható meg három típus:

Kódszerkesztő: A bal oldalon elhelyezkedő ablakrész betűtípusa, amibe a programkódot írhatjuk. Ennek a betűtípusnak érdemes valamilyen fix szélességű (monospaced) típust választani, pl.: consolas, courier new.

Animáció: A Lista nézet tetején megjelenő animációs rész. Ezt a betűt érdemes nagy méretűnek választani.

Ablak: A program összes többi része (menük, listák stb.).

1.3 A Prolan nyelv

Szintaktikai szempontból a nyelv alapja a C programozási nyelv, amellyel sok hasonlóságot mutat. Ilyen hasonlóság pl. hogy a kis és nagybetűt megkülönbözteti, a deklarációk alakja, a blokkok, függvények használata stb.. Egyéb szempontból a Prolan sokkal egyszerűbb és könnyebben tanulható mint a C, mivel nem általános célú programnyelv. Nincsenek változói, pointeri. Az egyetlen érték típusa a logikai típus, a szokásos alaptípusok (számok, string, tömb stb.) mind hiányoznak, és nem hozhatóak létre új típusok, vagyis nincs típuskonstrukció.

Az imperatív nyelvek lényege, hogy változókon végzett műveleteken keresztül módosítjuk a számítógép belső állapotát. Programozott nyelvtanok esetén ez úgy értelmezhető, hogy a programnak egyetlen változója van, az aktuális mondatforma, a műveletek pedig amikkel módosítjuk, a nyelvtani szabályok. Ilyen értelemben tehát a Prolan egy imperatív nyelv.

1.3.1 Hello World!

```
start S;  
term Hello, World;  
void main()  
{  
    S = Hello World;  
}
```

A fenti program deklarál egy startszimbólumot: S, és két terminális jelet: Hello és World. Valamint definiál egy main függvényt, melynek egyetlen sora az S jelet kicseréli a Hello World szóra. Ez a program tehát egy olyan generatív nyelvtant ír le, mely egyetlen szót generál: Hello World. (felkiáltójel nem szerepelhet nyelvtani jelben, ezért erről most le kellett mondanunk)

Általában a program törzse tartalmazhat:

- Makródefiníciókat.

- Globálisan látható szimbólumok deklarációit.
- Tartalmaznia kell pontosan egy **void main()** makródefiníciót.
- Fordítási direktívákat, kommenteket.

Nézzünk egy tanulságosabb példakódot, ami a 3 faktoriálisát „számolja ki”.

Példa

```
#pragma leftmost
#include "matek.pla"
start S;
void fact(nterm X, nterm N)
{
    X = N;
    all X = eps;
    N = X;
    while (N = eps) Szor(X, N);
}

void main()
{
    nterm X, N;
    S = X X N N N;
    fact(X, N);
}
```

A fenti program bemenetét az $S = X X N N N$; utasítással adjuk meg. Ebben 3db N szerepel, amit a **fact** eljárás úgy fog értelmezni, hogy az X szimbólum előfordulási darabszámát a mondatformában $3!$ -ra kell beállítania, miközben N darabszámát nullára csökkenti (vagyis a várt eredmény: $X X X X X X$). N kezdeti darabszáma zérus is lehet, hiszen $0! = 1$ értelmezhető (ez indokolja az eljárás feleslegesnek tűnő első három sorát).

Néhány szó az utasításokról: Az $X = N$ logikai kifejezés egy $X \rightarrow N$ nyelvtani szabály végrehajtásának felel meg. A kifejezés értéke **true**, ha szerepelt X az aktuális mondatformában, különben **false** (ebben az esetben most a program terminálna). Az **all** utasítás addig értékeli ki a mögötte álló logikai kifejezést, amíg annak értéke igaz, vagyis eredményét tekintve most kitörli a mondatformából az összes X -et. Az **eps** jelentése a görög ε (üres szó). A **while** utasítás a jól megszokott elől tesztelési ciklus. A **Szor** eljárás definíciója a `lib/math.pla` fájlban található, amit a kód elején az **#include** direktívával beszerkesztettünk. A két paraméterként átadott szimbólum darabszámát összeszorozza, ennyi darabot tesz vissza az első paraméterben szereplő szimbólumból, valamint változatlanul hagyja a második paraméter darabszámát. A paraméterek típusa **nterm**, ami a nemterminális jelek típusa.

Megfigyelhetjük, hogy a program futása szempontjából csak a szimbólumok előfordulási számának van jelentősége, az elhelyezkedésüknek nincs (vagyis gyakorlatilag olyan, mintha egy zsak struktúrán végeznénk műveleteket). A PNY programok jelentős része ebbe a típusba tartozik. Ilyen esetekben teljesítmény-szempontból érdemes bekapcsolni a **#pragma leftmost** direktívát, hogy legbal levezetéseket gyártson a program.

1.3.2 Fordítási direktívák, kommentek

A program egy C előfordítót használ, ezért a kommentezés szabályai és a használható fordítási direktívák megegyeznek a C-ben megszokottakkal (bővebben ld. (1)), kiegészítve néhány új direktívával:

- Egysoros komment: `//` jeltől a sor végéig.
- Többsoros komment: `/* szöveg.... */`
- **#include** <filename>: A filename fájl tartalmát beszerkeszti az utasítás helyére.
- **#define** <NAME>: Definiálja a NAME tokenet. Ld. még **#ifdef**, **#endif**. (1)
- **#pragma**: Minden Prolan-specifikus (vagyis nem C) fordítási direktíva **#pragma**-val kezdődik
(A pragma általában implementációfüggő fordítási direktívát jelöl. Ezeket az utasításokat a C előfordító változatlanul hagyja, és átadja őket a fordítóprogramnak.)
- **#pragma pgrammar|grammar|lindenmayer|pushdown|epushdown|turing**
Közi a fordítóprogrammal, hogy a forráskód milyen típusú nyelvtant/automatát ír le:
 - **pgrammar**: Programozott nyelvtan. Ez az alapértelmezett, elhagyható.
 - **grammar**: Közönséges nyelvtan. Hatása megegyezik **pgrammar**-ral, azzal a különbséggel, hogy bekapcsolja a „Zsákutcák elkerülése” funkciót. (ld. 1.2.2)
 - **lindenmayer**: Lindenmayer-rendszer. Hatása alapvetően megegyezik **pgrammar**-ral, tőle annyiban különbözik, hogy bekapcsolja a determinisztikus lépések elrejtését, a színek elrejtését Lista nézetben, valamint legbal levezetést állít be az összes nézet számára.
 - **pushdown**: Végállapottal felismerő veremautomata.
 - **epushdown**: Üres veremmel felismerő veremautomata.
 - **turing**: Turing-gép
- **#pragma leftmost**: Legbal levezetést állít be mindegyik nézet számára (Lista, Fa, Nyelv). Hatása megegyezik a Beállítások ablakban a „Nyelvtani szabály bal oldalát melyik pozícióra illessze?” opcióval, viszont annál magasabb prioritású. Vagyis ha a kódban szerepel ez a direktíva, a Beállításokban szereplő értéket ignorálja a program. Ugyanez vonatkozik az összes alábbi direktívára.
- **#pragma rightmost**: Legjobb levezetést állít be mindegyik nézet számára.
- **#pragma hide(determ)**: bekapcsolja a „Determinisztikus lépések elrejtése” funkciót. Ez az utasítás megegyezik az azonos nevű funkcióval a Beállítások panelen. **determ** helyett állhat:
 - **deadend**: Bekapcsolja a „Zsákutcák elkerülése” funkciót.
 - **all**: Minden lépést elrejt.
 - **color**: Elrejt a színeket a Lista és Fa nézetben.
A **determ**, **deadend** és a **color** hatása a kód egészére kihat, a **#pragma hide(all)** viszont kombinálva a **#pragma show(all)** utasítással kódrészletekre is használható (így csak az adott utasítások által generált lépéseket rejti el).
- **#pragma show(determ)**: A fenti utasítás ellentettje.
- **#pragma printmode**: Aktiválja a **print** utasításokat (ld.: 1.3.7).

- **#pragma position(list, left)**: Legbal levezetést állít be a Lista nézet számára. Megegyezik a Beállítások panel „Nyelvtani szabály bal oldalát melyik pozícióra illessze?” funkcióval.
 - **list** helyett állhat **tree** (vagyis Fa) vagy **lang** (Nyelv).
 - **left** helyett állhat **right** (jobb) és **all** (mind, ami a lista nézet esetén a véletlennek felel meg).
- **#pragma rule(list, left)**: Megadja, hogy a Lista nézetben, ha több szabály közül is lehet választani egy lépésben, akkor mindig az első szabályt válassza. (ld. 1.2.2, Szabály választás). **list** és **left** helyett ugyanazok állhatnak mint a fönti direktívánál. Erre a direktívára ritkán van szükség. Általában akkor, ha ki akarjuk kényszeríteni az automata determinisztikus működését.

Példák

```

1. all X = x; // minden X-et x-re cserél
2. /* all X = x;
   if (A = A) B = b; // egysoros komment is lehet benne
*/
3. #include "math.pla"
4. #pragma turing
5. #pragma position(tree, all)
6. #pragma rule(lang, right)

```

1. Egysoros komment. A // jel utáni rész nem kerül lefordításra.
2. Többsörös komment. Egyik utasítás sem kerül lefordításra.
3. A fordító beszerkeszti a direktíva helyére a math.pla fájl tartalmát.
4. Utasítja a fordítóprogramot, hogy Turing-gépként értelmezze a forráskódot.
5. A fa nézetben mindegyik pozícióra illeszt szabályt.
6. A nyelv nézetben ha egy lépésnél több szabály is alkalmazható, akkor mindig az utolsót választja.

1.3.3 Típusok

Függvények visszatérési értéke lehet:

- **bool**: Logikai
- **void**: Érték nélküli.

Szimbólumok típusai:

- **letter**: Nyelvtani jel. Altípusai:
 - **term**: Terminális jel.
 - **nterm**: Nemterminális jel. Altípusa:
 - **start**: Startszimbólum.

1.3.4 Deklarációk

Nyelvtani jeleket és függvényeket deklarálhatunk, változókat nem, mivel a programnak nincsenek változói. A deklarált nevek angol betűket és számjegyeket tartalmazhatnak (kezdődhet számmal, és állhat csak számból is).

Szintaxis

```
<deklaráció> ::= ("start" | "term" | "nterm") <név> ("," <név>)* ";"  
<név> ::= (<szám> | <betű>)*
```

Példák

```
1. start S;  
2. nterm 1, A, alma1, 6alma;  
3. term a, 2, alma2;
```

1. Deklarál egy startszimbólumot: S.
7. Deklarál négy nemterminális jelet: 1, A, alma1, 6alma.
8. Deklarál három terminális jelet: a, 2, alma2.

Megjegyzések

- A startszimbólum szokásos jele S.
- Terminális jeleket szokásosan a latin ábécé első kisbetűivel jelöljük: a, b, c, ...
- Nemterminális jeleket szokásosan a latin ábécé első nagybetűivel jelöljük: A, B, C, ...
- A számokat érdemes az utasítások felcímkézéséhez fenntartani, pl.: 1: S = a b c;
- Címkéket nem lehet deklarálni, felismerésük automatikus. A címkék használatát ld. az utasítások fejezetben (0).

1.3.4.1 Hatókör, láthatóság

Egy szimbólum hatóköre megegyezik a blokk hátralevő részével, amiben deklaráltuk.

Lehetséges globális szimbólumokat deklarálni, ezeket a program elején a blokkokon kívül kell elhelyezni.

Ha egy szimbólum neve megegyezik egy korábban deklarált szimbólumével, akkor elrejtí az. A generált nyelvtanban ezeket az azonos nevű szimbólumokat indexekkel különbözteti meg a program. Példa az elrejtésre és indexelésre:

Programozott Nyelvtan		Definíció	
ID	Szabály	σ	φ
0	$A \rightarrow a$	1	exit
1	$A1 \rightarrow b$	2	exit
2	$A2 \rightarrow c$	exit	exit

6. ábra: Példa elrejtésre és indexelésre.

1.3.5 Beépített konstansok

<i>Kulcsszó</i>	<i>Típus</i>	<i>Jelentés</i>
true	bool	Hamis logikai érték.
false	bool	Igaz logikai érték.
eps	term	Epsilon görög betű. Az üres szó jele.
_	term	"blank", az üres karakter T-gép esetén
exit	label	Elutasító állapot címkéje, terminálja a programot.
accept	label	Elfogadó állapot címkéje, terminálja a programot.

1.3.6 Logikai kifejezések

Mivel a Prolan-ban az egyetlen érték típus a **bool**, csak logikai operátorok vannak, melyek a következők:

<i>Jel</i>	<i>Jelentés</i>	<i>Precedencia</i>	<i>Asszociativitás</i>
=	Helyettesítés	5	nem asszociatív, bináris
&&	Lusta és	4	bal, bináris
!	Negáció	3	jobb, unáris
 	Lusta vagy	2	bal, bináris
 	Elágazó vagy	1	bal, bináris

PNY esetén a „helyettesítés” az aktuális mondatformán hajt végre egy adott nyelvtani szabálynak megfelelő helyettesítést. A szabály bal oldala a bal oldali operandus, jobb oldala a jobb oldali. Értéke **true** ha a helyettesítés végrehajtható, **false** ha nem. VA esetén egy veremműveletet ír le: a bal oldali operandust kiveszi a veremből (amennyiben az olvasható a verem tetején), és a jobb oldali operandust írja a helyére. A példákban a PNY esetét fogjuk alapul venni, a VA-król részletesebben lásd a 0 fejezetben.

Az „elágazó vagy” kiértékelése nemdeterminisztikus: véletlenszerűen végrehajtja valamelyik operandusát, értéke megegyezik a kiválasztott operandus logikai értékével. A nem kiválasztott operandus nem hajtódik végre, nem befolyásolja az eredményt.

Szintaxis

```

<helyettesítés> ::= <mondatforma> "=" <mondatforma>
<és> ::= <logk> "&&" <logk>
<vagy> ::= <logk> "||" <logk>
<negáció> ::= "!" <logk>
<elágazó vagy> ::= <logk> "|" <logk>
<logk> ::= <helyettesítés> | <és> | <vagy> | <negáció>
          | <elágazó vagy> | <loghívás> | "true" | "false"

```

Példák

1. $A = B \wedge a = b$
2. $(A = a) \ \&\& \ (B = b)$
3. $(A = a) \ || \ (B = b)$
4. $! A = a$
5. $(A = a) \ || \ (B = b) \ || \ (C = c)$

1. A-t cseréli a b B-re. Értéke **true**, ha volt A a mondatformában, egyébként **false**.
9. A-t a-ra cseréli, ha sikerült akkor B-t b-re cseréli. Ha valamelyik csere nem sikerült, a kifejezés értéke **false**, különben **true**.
10. A-t a-ra cseréli, ha sikerült akkor a kifejezés értéke **true**, különben B-t b-re cseréli. Ha ez sikerült a kifejezés értéke **true**, egyébként **false**.
11. A-t a-ra cseréli, ha sikerült akkor a kifejezés értéke **false**, különben **true**.
12. Az alábbi három lehetőség közül véletlenszerűen végrehajtja valamelyiket:
 - A-t a-ra cseréli, ha sikerült akkor a kifejezés értéke **true**, különben **false**.
 - B-t b-re cseréli, ha sikerült akkor a kifejezés értéke **true**, különben **false**.
 - C-t c-re cseréli, ha sikerült akkor a kifejezés értéke **true**, különben **false**.

1.3.6.1 Logikai kifejezések mint utasítások

Minden logikai kifejezésből alkotható utasítás egy pontosvesszővel:

```
<utasmag> ::= <logk> ";" | ...
```

Jelentése: a kifejezés végrehajtódik. Ha értéke **true**, a program futása folytatódik a következő utasításon. Ha értéke **false**, a program terminál.

Példa

```
(A = a) && (B = b); C = c;
```

A-t a-ra cseréli, ha sikerült akkor B-t b-re cseréli. Ha valamelyik csere nem sikerült, a program terminál, különben folytatódik a következő utasításon, vagyis C-t c-re cseréli.

1.3.7 Utasítások

Az alábbiakban felsoroljuk azokat az utasításokat, melyek mindhárom alkalmazási területen használhatóak. A T-gép és VA specifikus utasításokat ld. a 1.4 fejezetben.

- **if** (<feltétel>) <utasítás>
Ha a feltétel igaz, végrehajtja az utasítást.
- **if** (<feltétel>) <utasítás1> **else** <utasítás2>
Ha a feltétel igaz, végrehajtja utasítás1-et, különben végrehajtja utasítás2-t.
- **while** (<feltétel>) <utasítás>
Amíg a feltétel igaz, végrehajtja az utasítást.
- **all** <feltétel> ;
Amíg a feltétel igaz, végrehajtja. (Hatása megegyezik **while** (<feltétel>); utasításával)
- **try** <feltétel>;
Kiértékeli a feltételt. Az utasítás feladata, hogy megakadályozza hamis érték esetén a program terminálását. Hatása megegyezik az **if** (<feltétel>); utasítással (a pontosvessző előtt egy utasítás hiányzik, ami így tkp. egy skipnek felel meg).

- **goto** <címke1>, <címke2>, ... , <címken>;
Véletlenszerűen ugrik a felsorolt címkék valamelyikére.
- <címke1>, <címke2>, ... , <címken>;
Hatása megegyezik az előbbivel, vagyis a **goto** kulcsszó elhagyható.
- **print**
Ha a kódban nem szerepel a **#pragma printmode** direktíva, akkor a fordító ignorál minden **print** utasítást. Ha szerepel, akkor minden **print** utasításhoz létrejön egy olyan művelet, mely a Lista nézetben megjeleníti az aktuális mondatformát (ill. szalagot, veremtartalmat). Megj.: a **#pragma hide(all)** nincs hatással a **print**-re, vagyis azokat nem rejti el. Az említett két direktíva és a **print** megfelelő használatával hatékonyan befolyásolhatjuk, hogy mely lépések kerüljenek fel a Lista nézet soraiba.

Szintaxis

```

<if> ::= "if" "(" <logk> ")" <utasítás> ("else" <utasítás>)?
<while> ::= "while" "(" <logk> ")" <utasítás>
<all> ::= "all" <logk> ";";
<try> ::= "try" <logk> ";";
<goto> ::= ("goto" <címkék>) | <címkék> ";";
<címke> ::= (<számjegy> | <betű>)+
<címkék> ::= <címke> ("," <címke>)*
<utasmag> ::= <if> | <while> | <all> | <goto> | <hívás> | <logk>";"
<utasítás> ::= (<címke> ":")? utasmag

```

Példák

```

1. if (A = a) B = b;
2. while (A = a) B = b;
3. all A = a;
4. try A = a;
5. goto lab1, lab2;
6. lab1, lab2;
7. if (A = a) 1, 2; else 3, exit;
8. lab1: A = a;
   2: B = b;

```

1. A-t a-ra cseréli. Ha sikerült akkor B-t b-re cseréli.
2. Amíg az A→a csere végrehajtható, B-t b-re cseréli.
3. Végrehajtja a feltételt amíg az igaz, vagyis minden A-t a-ra cserél.
4. Megpróbálja végrehajtani az A→a cserét. Ha nem sikerült, akkor sem terminál.
5. Véletlenszerűen a lab1 vagy a lab2 címkére ugrik.
6. Ugyanaz mint a fönti, vagyis a **goto** elhagyható.
7. A-t a-ra cseréli, ha sikerült, az 1 vagy a 2 címkére ugrik. Ha nem, akkor terminál, vagy a 3 címkére ugrik.
8. Címkével ellátott utasítások.

1.3.8 Blokkok

A nyelvben háromféle blokkutasítás van:

„**Szekvenciális**”: A blokk utasításai egymás után kerülnek végrehajtásra. (ez megfelel az imperatív nyelvekben megszokott blokkutasításoknak). Jele: { }

„Elágazó”: A blokk utasításai közül véletlenszerűen kiválaszt egyet, azt végrehajtja, majd kilép a blokkból. Jele: < >

„Mátrix”: A blokk utasításai közül újra és újra választ egyet véletlenszerűen, amíg a program nem terminál. Egyfajta végtelen ciklusként viselkedik, amiből csak terminálással lehet kilépni. Ez az utasítás hasznos közönséges nyelvtanok és mátrixnyelvtanok modellezésére (ld. 1.4). Jele: []

Megjegyzés: mindhárom szerepelhet makró blokkjaként is, vagyis pl. `void main() [S = a;]` egy érvényes makródefiníció (ld. 1.4 fejezet).

Szintaxis

```
<szekvblokk> ::= "{" <utasítás>* "}"  
<forkblokk> ::= "<" <utasítás>* ">"  
<mátrix> ::= "[" <utasítás>* "]"
```

Példák

```
1. { A = a; B = b; }  
2. < A = a; B = b; >  
3. [ A = a; B = b; ]  
4. [ A = a; { B = b; C = c; } ]
```

1. A-t a-ra cseréli (ha nem sikerült, akkor terminál), majd B-t b-re (ha nem sikerült, akkor terminál).
2. Az alábbi két lehetőség közül választ egyet és végrehajtja.
 - A-t a-ra cseréli. Ha nem sikerült, akkor terminál.
 - B-t b-re cseréli. Ha nem sikerült, akkor terminál.
3. Az alábbi két lehetőség közül választ egyet és végrehajtja. Ha a végrehajtás sikerült újra válasz, ha nem akkor terminál.
 - A-t a-ra cseréli.
 - B-t b-re cseréli.
4. Az alábbi két lehetőség közül választ egyet és végrehajtja. Ha a végrehajtás sikerült újra válasz, ha nem akkor terminál.
 - A-t a-ra cseréli.
 - B-t b-re cseréli, majd C-t c-re cseréli

1.3.9 Makrók

A Prolan makrók néhány dologban különböznek a C előfordító makróitól, de a C függvényeitől is. Az alábbiakban összefoglaljuk hasonlóságokat és különbségeket:

Makrószerű tulajdonságok:

- A hívás feloldása fordítási időben történik. Ennek alapvető oka az, hogy nincs műveleti memória amiben futás közben lokális változókat tárolhatnánk, és elkülöníthetnénk a hívó programrésztől. Ugyanezen okból nem tárolhatnánk a hívási vermet sem.
- Rekurzív hívás engedélyezett, de ciklikus rekurzió nem. Ez a fordítási időben való feloldás következménye, ciklikus rekurzió végtelen ciklust okozna a fordítás során.
- A deklaráció és a definíció nincs különválasztva, nincs elődeklaráció. Ez a szabály biztosítja a ciklikus rekurzió tiltását.

Függvényszerű tulajdonságok:

- A szintaxis megegyezik a C függvényekével, szemben a nehezebben használható C makrók szintaxisával.
- A paraméterek lokális „változóként” viselkednek, nem termhelyettesítés történik.
- A makró **bool** vagy **void** típusú lehet, vagyis lehet visszatérési értéke, amiket a szokásos **return** utasítással adhatunk meg.

Szintaxis

```
<függvdef> ::= ("void" | "bool") | <név> <paramdeks> <blokk>
<paramdeks> ::= "(" (<paramdek> ("," <paramdek>)*)? ")"
<paramdek> ::= ("term" | "nterm") <név>
<hívás> ::= <név> "(" (<név> ("," <név>))* ")" ";"
```

Példák

```
void main() { S = a; }
bool f(nterm X)
{
    X = a;
    if (Y = b) return Z = b;
    return false;
}
```

1.4 Speciális nyelvtanok, automaták programozása

A Prolan segítségével közvetlenül írhatunk le programozott nyelvtant, nemdeterminisztikus Turing-gépet vagy veremautomatát. Valamint ezek felhasználásával különféle egyéb nyelvtan és automata típus szimulálható (ezek sora remélhetőleg bővülni fog, a felhasználó is kísérletezhet új alkalmazások keresésével). Az alábbi fejezetekben bemutatom azokat az utasításokat és programozási technikákat, melyek a különböző speciális felhasználási területekhez szükségesek.

1.4.1 Közösnyelvi nyelvtanok

A **#pragma grammar** direktívával utasíthatjuk a fordítóprogramot, hogy a kódot közösnyelvi nyelvtanként értelmezze. Ez mindössze annyi különbséget jelent az alapbeállításhoz képest (**#pragma pgrammar**), hogy bekapcsolja a „zsákutcák elrejtése” opciót. Ennek értelmét az alábbi példán láthatjuk. Tekintsük a helyes zárójelezések nyelvét generáló közösnyelvi nyelvtant, és szimuláljuk PNY-al:

címke	szabály	σ	φ
f_1	$S \rightarrow SS$	f_1, f_2, f_3	<i>exit</i>
f_2	$S \rightarrow (S)$	f_1, f_2, f_3	<i>exit</i>
f_3	$S \rightarrow \varepsilon$	f_1, f_2, f_3	<i>exit</i>

Látható, hogy a σ oszlop tartalma minden sorban megegyezik: tartalmazza az összes nyelvtani szabály címkéjét. Ennek jelentése az, hogy egy adott szabály végrehajtása után mindenféle megkötés nélkül bármelyik másik szabályt alkalmazhatjuk. A φ oszlopban mindenütt *exit* szerepel, vagyis ha az aktuális szabályt nem tudtuk alkalmazni, akkor minden esetben terminál a levezetés.

A fenti alakú PNY-ok tehát minden lépésben minden lehetséges szabályt kipróbálnak, és a sikertelen próbálkozások az *exit* címkén terminálnak. Ennek eredményeként egy véletlenszerűen választott PNY levezetés nagy valószínűséggel az *exit* címkén terminál akkor is, amikor pedig a levezetés még folytatható volna, mert választható alkalmas szabály. Ez a viselkedés természetesen megfelel a PNY definíciójának, de abban az esetben ha a cél közönséges nyelvtan modellezése, jó lenne elkerülni. Ebben segít a „zsákutcák elkerülése” beállítás, amely egy lépést előre tekintve elkerüli az *exit* címke választását. Pontosabban: nem választ olyan címkét, amely után mindenképpen az *exit*-et kell választani, csak akkor, ha nincs más választás.

A blokkutasítások között már találkoztunk a „mátrix” blokkal (1.3.8). Látható, hogy ennek viselkedése pont megfelel annak, amire közönséges nyelvtanok esetén szükségünk van: szabályok egy halmazából ismételten addig választ, amíg a program nem terminál. Példaként tekintsük a fenti helyes zárójelezések nyelvét generáló nyelvtant leíró kódot.

Példa

```
#pragma grammar
start S;
term a, b; // a = "(", b = ")"
void main()
[
    S = S S;
    S = a S b;
    S = eps;
]
```

1.4.2 Mátrixnyelvtanok

A mátrixnyelvtanok és Lindenmayer-rendszerek matematikai definícióval itt nem foglalkozunk (részletesen ld. (2)), csak nagyon röviden és vázlatosan illusztrálom a fogalmak jelentését a példafájlok használhatósága érdekében.

Mátrixnyelvtan alatt olyan nyelvtant értünk, melyben a nyelvtani szabályok szekvenciákba vannak rendezve. A szekvenciákon belül a szabályok végrehajtási sorrendje rögzített, viszont a szekvenciák közötti sorrend tetszőleges. Valamint a szekvenciák tranzakcióként viselkednek, vagyis vagy végrehajtjuk az elejétől a végéig, vagy el sem kezdjük. Ha már ismerjük a Prolog néhány utasítását, akkor az alábbi példakódból azonnal világossá válik a mátrixnyelvtanok működése.

Példa: Tekintsük a $[S \rightarrow ABC]$, $[A \rightarrow Aa, B \rightarrow Bb, C \rightarrow Cc]$, $[A \rightarrow a, B \rightarrow b, C \rightarrow c]$ mátrixnyelvtan kódját.

```
start S;
nterm A, B, C;
term a, b, c;
void main()
[
    S = A B C;
    { A = a A; B = b B; C = c C; }
    { A = a; B = b; C = c; }
]
```

A mátrixnyelvtanok fordítása mindenben megegyezik a programozott nyelvtanokéval. Arra kell csak ügyelnünk, hogy ha „tisztán” mátrixnyelvtant akarunk létrehozni, akkor csak bizonyos utasításokat használhatunk: értékadás, mátrix blokk, és a try utasítás (pontosított szabályokhoz).

Pontosított szabály alatt olyan szabályt értünk, melyet ha alkalmazni akarunk az aktuális mondatformára, és ez sikertelen, a levezetés nem ér véget, hanem folytatódik a szekvencia következő szabályán. Pontosított szabályt a **try** utasítás segítségével hozhatunk létre, pl. az $A \rightarrow a$ szabálynak a **try** $A = a$; utasítás felel meg.

1.4.3 Lindenmayer-rendszerek

Lindenmayer-rendszer alatt leegyszerűsítve és vázlatosan a következőt értjük:

Tekintsünk egy A ábécét, valamint egy $\sigma: A \rightarrow 2^{A^*}$ függvényt (σ tehát betűkhöz szavak egy halmazát rendel). A levezetés egy lépése abból áll, hogy a mondatforma minden egyes $a \in A$ betűjét átírjuk egy $u \in \sigma(a)$ szóra.

Példa: $MISS_3$ L-rendszer

Ábécé: $\{x\}$; átíró függvény: $\sigma(x) = \{\varepsilon, xx, xxxxx\}$

Egy lehetséges levezetés: $x \rightarrow xx \rightarrow xxxxxxxx \rightarrow \varepsilon$

(a második lépésben az egyik x -et xx -re, a másikat $xxxxx$ -re cseréltük. Az utolsó lépésben minden egyes x -et ε -ra.)

Generált nyelv: $L(MISS_3) = \{x^n | n \in \mathbb{N}_0 \setminus \{3\}\}$

Az iménti nyelvet a következő Prolan kód generálja:

```
#pragma lindenmayer
start X;
nterm Y;
term x;
void main()
[
    {
        all X = eps | X = Y Y | X = Y Y Y Y Y;
        all Y = X;
    }
    all X = x;
]
```

A **#pragma lindenmayer** direktívával utasítottuk a fordítóprogramot, hogy a kódot Lindenmayer-rendszerként értelmezze. Ez annyiban különbözik az alapbeállítástól, hogy bekapcsolja a determinisztikus lépések elrejtését, a színek elrejtését Lista nézetben, valamint legbal levezetést állít be az összes nézet számára.

A fenti példában az „elágazó vagy”, vagyis a „|” operátort használtuk az átírási szabály megvalósítására. Emlékeztetőül: az $A \mid B$ logikai kifejezés véletlenszerűen vagy A-t, vagy B-t értékel ki, és a kifejezés értéke megegyezik ezzel az értékkel. Az **all** utasítással együtt alkalmazva eredményül azt kapjuk, hogy az utasítás a mondatformában szereplő összes X-re alkalmazza a felsorolt szabályok valamelyikét, míg végül nem marad X a mondatformában.

Fent azért nem alkalmazhattunk $X = X X$ és $X = X X X X X$ szabályokat, mert ebben az esetben a program nem tudná megkülönböztetni hogy melyek az eredetileg szereplő X-ek, és melyek az újak. Így pl. a fenti példában végtelen ciklust kapnánk, de más esetekben is rossz működéshez vezetne. Ezért kellett egy új Y szimbólumot bevezetnünk, mely X helyettesének tekinthető. Miután minden X-et lecseréltünk, a helyetteseket vissza kell cserélnünk X-ekre, erre szolgál az **all** $Y = X$; utasítás.

A main() makró blokkja egy mátrix blokk, ami végtelen ciklusban véletlenszerűen választ a fent részletezett lépés, ill. az **all** $X = x$; utasítás közül. Ez utóbbi minden nemterminálist terminálisra cserél, vagyis előállítja a $MISS_3$ nyelv egy elemét, és terminálja a programot.

1.4.4 Automaták

A Prolan közvetlenül kétféle automatát támogat:

- ϵ -átmenetes nemdeterminisztikus Turing-gép. Ez az automata alesetként tartalmazza a determinisztikus ill. ϵ -mentes változatokat is, valamint könnyen modellezhető segítségével véges automata.
- (nemdeterminisztikus) Veremautomata. Változatai: üres veremmel felismerő, végállapottal felismerő.

A nyelv tervezésekor elsődleges cél volt, hogy az utasításkészlet minél kisebb és egyszerűbb legyen, hogy ugyanazokkal az utasításokkal lehessen a különböző automata variációkat leírni. Az automaták működtetése/futtatása szempontjából szintén cél volt a takarékoság és egységesség. Ennek következményeként a Prolan által generált automaták nem mindig felelnek meg pontosan a matematikai definícióknak, viszont ezeket is megtekinthetjük a munkalapok közepén elhelyezkedő „Definíció” fülre kattintva. Az eltérések:

- Minden ábécé tartalmazza az **nterm** típusú _ (blank) karaktert. Veremautomatánál és véges automatánál ennek a karakternek az olvasása a bemenet végét jelöli.
- Minden automatának van előre definiált **exit** és **accept** állapota. A különböző automaták definíció szerinti megállási feltételeit (van-e végállapot, végig kell-e olvasni a bemenetet stb.) a blank karakter olvasásával és az **exit/accept** használatával lehet szimulálni.

Az automaták megadásánál használhatóak mindazok az utasítások, vezérlési szerkezetek, makróhívások mint a nyelvtanoknál. Az utasítások ügyes megválasztásával a kód sokkal rövidebb és áttekinthetőbb lehet, mint a kész automatát leíró táblázat. Ugyanakkor az automaták esetén nagyon könnyű érvénytelen, vagy hibásan működő kódot megadni, ezért érdemes csak bevált módszereket alkalmazni. A példafájlokban mintát láthatunk arra, milyen programozói stílussal lehet egyszerűen és megbízhatóan T-gépet és veremautomatát megadni. Erre egyik lehetőség a **state** utasítás, aminek segítségével az automaták állapotait és átmeneteit gyakorlatilag explicit módon írhatjuk le. Az automaták közös utasításai:

- **input(<szó>)**
VA esetén a <szó> adja a bemenetet, T-gép esetén a szalagra másolja a <szó>-t, és az első karakterére állítja a fejet. A <szó> alakja: **term** típusú jelek szóközzel elválasztva.

- **read(<term>)**
VA esetén a bemenetről próbál <term>-et olvasni, az olvasás sikere az utasítás visszatérési értéke. T-gép esetén a szalag fej alatti részét próbálja olvasni.
- (<term>)
Megegyezik a fentivel (vagyis a **read** elhagyható).
- **state {**
 <feltétel1>: <utasítás1>
 <feltétel2>: <utasítás2>
 ...
 <feltételn>: <utasításn>
 else <utasításelse>
}

Az automata egy állapotának leírására szolgál. A program kiértékeli <feltétel1>-et, és ha igaz, végrehajtja <utasítás1>-et. Ha a feltétel hamis volt, megvizsgálja <feltétel2>-t stb.. Ha ugyanaz a feltétel többször is szerepel, akkor véletlenszerűen hajtja végre valamelyik előfordulást. A feltétel általában egyetlen jel olvasását jelenti, de nem szükségszerűen. Az **eps** olvasása mindig sikeres, ezekkel hozhatunk létre ε -átmeneteket. Ha egyik feltétel sem teljesül, végrehajtja <utasításelse>-t. Az **else** ág elhagyható, ez megfelel annak, mintha **else exit;** állna ott (ez a működés a programozás szempontjából szokatlan lehet, de pontosan megfelel az automaták megadásával kapcsolatos konvencióknak).

A fordítóprogram a **state** utasítást valójában **if-else** ágak láncaként értelmezi. Erre az utasításra külön adunk majd példát T-gép és VA esetén.

Példák

```
1. input(a a b b);
2. read(a)
3. a
4. (a);
5. a || b
```

1. VA esetén beállítja bemenetnek az a a b b szót. T-gép esetén rámásolja a szalagra, és az első karakter fölé állítja a fejet.
2. Ha a bemenet/szalag aktuális betűje a, akkor **true**, különben **false**.
3. Ugyanaz mint fent, vagyis a **read** elhagyható.
4. Ha a bemeneten a-t olvas, akkor folytatja a végrehajtást, különben terminálja a programot. Ebben az esetben (vagyis amikor az olvasás önálló utasításként szerepel és nem részkifejezésként) zárójelbe kell tenni a szimbólumot. Erre azért van szükség, mert a sima a; könnyen összetéveszthető volna az ugró utasítással, aminek ugyanez az alakja.
5. Ha a bemeneten a-t vagy b-t olvas, az értéke **true**, különben **false**.

1.4.5 Turing-gép

A programunkban néhány kivétellel használhatjuk a korábban felsorolt típusokat és utasításokat, bizonyos esetekben megváltozott jelentéssel. Az eltérések:

- A **start** típusnak T-gép esetén nincs jelentése, az így definiált szimbólumot **nterm** típusúnak tekinti a fordító.
- A **term**-ként deklarált jelek alkotják a „bemenő jelek ábécé-jét”.
- Az **nterm**-ként deklarált jelek hozzáadódva a „bemenő jelek ábécé-jéhez” megadja a szalagábécé elemeit. A `_` karakter (blank) alapértelmezés szerint része ennek az ábécének.
- Nem használható a helyettesítés művelet. (vagyis az „=” operátor)

1.4.5.1 *Specifikus utasítások*

- **left(<term>);**
A `<term>`-et kiírja a szalagra, majd balra lépteti a fejet.
- **left;**
Ha a szalagon **term** típusú betű olvasható, akkor visszaírja a szalagra ezt a betűt, azután balra lép. Vagyis eredményét tekintve egyszerűen balra lép, függetlenül a szalag tartalmától. Ha az olvasott betű **nterm** típusú, akkor a program terminál. (a tapasztalatok szerint ez a működés felel meg leginkább a gyakorlatnak, és eredményezi a legkevesebb redundáns állapotátmenetet)
- **right(<term>);**
A `<term>`-et kiírja a szalagra, majd jobbra lépteti a fejet.
- **right;**
Jobbra lép, függetlenül a szalag tartalmától, avagy terminál. (ld. **left** utasítás)
- **stand(<term>);**
A `<term>`-et kiírja a szalagra, a fej állva marad.
- **stand;**
A fej állva marad.

Példa

```
q0: state
{
    a: { left; q1; }
    b || c: { right(d); q2; }
    eps: { right(e); q3; }
}
```

A következő lehetőségek közül véletlenszerűen végrehajtja valamelyiket:

- Ha a szalagon a fej alatt „a” olvasható, a fej balra lép, és ugrik a q1 címkére.
- Ha b vagy c olvasható, d-t ír a szalagra, jobbra lép majd q2 címkére ugrik.
- Bármilyen is olvasható a szalagon, e-t ír a szalagra, jobbra lép és ugrik q3-ra.
- Ha a szalagon olyan jel olvasható, mely nem szerepel a többi olvasó utasításban (pl. e), akkor az automata elutasító állapotba ugrik, és terminál.

Összefoglalva: a fenti kód végeredményben a Turing-gép egy q0 állapotát írja le, melynek átmenetei:

$$\begin{aligned} \delta(q_0, a) &= (q_1, a, \text{left}) & \delta(q_0, b) &= (q_2, d, \text{right}) \\ \delta(q_0, c) &= (q_2, d, \text{right}) & \delta(q_0, \varepsilon) &= (q_3, e, \text{right}) \end{aligned}$$

Valamint: minden egyéb esetben a program terminál, vagyis az állapotnak nincs több átmenete.

A fenti példában a **state** utasítás címkéje egy egyszerű utasításcímke, amit viszont érdemes magával az állapottal azonosítani.

1.4.5.2 Állapotok és állapotátmenetek

A Turing-gép minden állapotátmenete három műveletet tartalmaz: olvasás, írás, léptetés. A Prolan-ban az írás és a léptetés csak együtt hajtható végre (ez a gyakorlat szempontjából nem jelent valódi megszorítást). A fordítóprogram feladata megtalálni az összetartozó olvasó ill. író/léptető műveleteket, és ezekből meghatározni az automata állapotait és átmeneteit.

Minden átmenetet pontosan egy író/léptető művelet zár le (**left**, **right** vagy **stand**), de tetszőleges számú olvasó műveletet tartalmazhat. Néhány példa átmeneteket definiáló kódra:

```
1. (a); left(b);
2. (a); left;
3. (a); left; left;
4. (a || b); left(d);
5. (a || b); left;
6. (a && b)
7. if (! a) { (b || c); left; }
8. while (! _) right;
9. q1: if (a) q2;
   q2: if (a) q1;
10. q1: if (a) { stand; q2; }
    q2: if (a) { stand; q1; }
```

1. $\delta(q, a) = (q, b, \leftarrow)$
2. $\delta(q, a) = (q, a, \leftarrow)$
3. $\delta(q1, a) = (q2, a, \leftarrow)$
 $\delta(q2, x) = (q3, x, \leftarrow)$, ahol x felveszi az összes **term** típusú szimbólumot.
4. $\delta(q1, a) = (q2, d, \leftarrow)$, $\delta(q1, b) = (q2, d, \leftarrow)$
5. $\delta(q1, a) = (q2, a, \leftarrow)$, $\delta(q1, b) = (q2, b, \leftarrow)$
6. Az $(a \ \&\& \ b)$ tkp. egy **false** utasításnak felel meg, hiszen az automata nyilvánvalóan nem olvashat egyszerre két különböző szimbólumot.
7. $\delta(q1, b) = (q2, b, left)$, $\delta(q1, c) = (q2, c, left)$
8. $\delta(q1, x) = (q2, x, right)$, ahol x tetszőleges **term** szimbólum, kivéve a „_”-t.
9. Ez a kód érvénytelen. Látható, hogy ha a szalagon „a” olvasható, akkor a gép végtelen ciklusba esik. A Prolan esetén ez a végtelen ciklus már a fordítás szakaszában létrejön, mivel a fordító az olvasó utasításokat nem képes összepárosítani író/léptető műveletekkel. Ez a működés tkp. elvárható, hiszen a kódhoz éppen az említett okból nem rendelhető Turing-gép.
10. Ez a kód csak kevésben különbözik az előzőtől, de már érvényes (habár a végtelen ciklus ezúttal a működés során jelentkezik majd).

$$\delta(q1, a) = (q2, a, stand), \quad \delta(q2, a) = (q1, a, stand)$$

1.4.5.3 Példaprogram

A T-gép fordításának részleteit legjobban a példafájlok tanulmányozásával érthetjük meg. A teljesség kedvéért viszont álljon itt egy komplett T-gépet leíró kód, amely pontosan azokat a 0-1 sorozatokat fogadja el, melyekben páros számú 1-es található.

```
#pragma turing
term 0, 1;
void main()
{
    input(0 0 1 1 0);
q0: state
{
    0: { right; q0; }
    1: { right; q1; }
    _: accept;
}
q1: state
{
    0: { right; q1; }
    1: { right; q0; }
}
}
```

Állapotok halmaza: $\{q_0, q_1, accept, exit\}$

Kezdőállapot: q_0

Elfogadó állapot: $accept$

Elutasító állapot: $exit$

Bemenő jelek ábécéje: $\{0, 1\}$

Szalagábécé: $\{0, 1, _\}$

Átmenetek:

$\delta(q_0, 0) = (q_0, 0, \rightarrow)$ $\delta(q_0, 1) = (q_1, 1, \rightarrow)$ $\delta(q_0, _) = (accept, _, \rightarrow)$

$\delta(q_1, 0) = (q_1, 0, \rightarrow)$ $\delta(q_1, 1) = (q_0, 1, \rightarrow)$

1.4.6 Véges automata

A Prolog Turing-gép segítségével képes a véges automaták modellezésére: Az automata bemenetét egy T-gép bemenetének tekintjük, a bemenet első karakterére állítjuk a T-gépet, majd jobbra lépegetve egyenként beolvassuk a bemenetet, és végrehajtjuk a véges automata átmeneteit. Az automata akkor fogadja el a bemenetet, ha a bemenet végigolvasása után (vagyis amikor már $_$ karaktert olvas) elfogadó állapotban van.

Az **exit** címke nem használható, mivel egy véges automata nem terminálhat azelőtt, hogy végigolvasta a bemenetet. Ha mégis ilyen működést akarunk megvalósítani, arról a felhasználónak kell gondoskodnia (pl. „csapda” állapotokkal).

A fentieknek megfelelően a tennivalók:

- Használjuk a **%pragma turing** fordítási direktívát
- Minden állapotváltás előtt adjunk ki egy **right;** utasítást.

- Minden elfogadó állapothoz adjunk hozzá egy **_**: **accept**; átmenetet (ill. ha nem **state** utasításokkal dolgozunk, ennek megfelel egy **(_)**; **accept**; szekvencia is). Egyéb helyen ne használjunk **accept** címkét, valamint sehol se használjunk **exit**-et.

A 1.4.5.3 fejezetben leírt T-gép egyben tekinthető véges automatának is, és megfelel a fent leírt szabályoknak.

1.4.7 Veremautomata

- VA esetén a **start** típussal a verem kezdőszimbólumát adhatjuk meg.
- A **term** típusú jelek adják a bemenő jelek ábécéjét, az **nterm** típusúakkal kiegészítve megkapjuk a veremábécét.
- A helyettesítés művelet jelentése eltér a PNY-hez képest. Alakja: **<betű>=<szó>**. A **<betű>**-t leolvassuk a verem tetejéről, és helyére a **<szó>**-t írjuk. Igaz értékkel tér vissza, ha a szabály végrehajtható, vagyis a verem tetején **<betű>** volt, különben hamissal. A **<szó>** verembe írását úgy kell értelmezni, hogy a szó betűit jobbról balra haladva egymás után betesszük a verembe, vagyis a szó első betűje lesz végül a verem tetején.
- Az **exit** címke nem használható, mivel egy VA nem terminálhat azelőtt, hogy végigolvasta a bemenetet. Ha mégis ilyen működést akarunk megvalósítani, arról a felhasználónak kell gondoskodnia (pl. „csapda” állapotokkal). Hasonló okból az **accept** csak speciális esetekben használható (ld. megállási feltételek).

Példák

```
1. a = a b c;
2. (a || b) && (c = e)
3. a && c = d; e = f;
4. q0: state
   {
     a:      { c = d; q1; }
     b || c: { e = f; q2; }
     eps:    { a = b; q3; }
   }
```

1. A verem tetején lévő *a* helyére *a b c*-t ír. Az írás sorrendje *c, b, a*, vagyis a verem tetején végül *a* lesz.
2. Ha a bemenet aktuális betűje *a* vagy *b*, akkor a verem tetején lévő *c*-t lecseréli *e*-re. Ha a bemenet nem *a* vagy *b*, ill. ha a veremtető nem *c*, akkor **false** értékkel tér vissza, és nem csinál semmit.
3.
 - Ha a bemeneten *a*-t olvas, akkor ellenőrzi a veremtetőt, különben a program terminál. Ha a veremtető *c*, akkor *d*-re cseréli (majd folytatja a köv. ponton), különben a program terminál.
 - Ha a veremtető *e*, akkor kicseréli *f*-re, különben a program terminál.
4. Ha a bemeneten *a*-t olvas, ellenőrzi a veremtetőt: ha a veremtető *c*, lecseréli *d*-re, majd ugrik a *q1* címkére. Hasonlóan a többi átmenetre is. Összefoglalva: a fenti kód végeredményben a veremautomata egy *q0* állapotát írja le, melynek átmenetei:

$$\begin{aligned} \delta(q_0, a, c) &= (q_1, d) & \delta(q_0, b, e) &= (q_2, f) \\ \delta(q_0, c, e) &= (q_2, f) & \delta(q_0, \varepsilon, a) &= (q_3, b) \end{aligned}$$

Valamint: minden egyéb esetben a program terminál, vagyis az állapotnak nincs több átmenete.

1.4.7.1 Megállási feltételek

A megállási feltétel szempontjából kétféle veremautomatát támogat a program:

Végállapottal elfogadó: Miután az automata végigolvasta a bemenetet (ami a Prolan esetén annyit tesz, hogy `_` jelet olvas), ha elfogadó állapotban van, akkor elfogadja a bemenetet, különben elutasítja. Ezt a viselkedést úgy tudjuk szimulálni, hogy az elfogadó állapotokhoz hozzáveszünk egy `_`: **accept**; utasítást. (ez megegyezik a véges automatáknál leírtakkal). Végállapottal elfogadó veremautomata esetén a **#pragma pushdown** fordítási direktívát kell használnunk.

Üres veremmel elfogadó: Miután az automata végigolvasta a bemenetet, ellenőrzi, hogy a verem tartalma üres-e (semmi nem szerepelhet benne, még a verem kezdőszimbólum sem). Ha üres, elfogadja a bemenetet, ha nem, elutasítja. Ezt a viselkedést úgy tudjuk szimulálni, hogy **#pragma epushdown** direktívát használunk, valamint nem használjuk az **accept** címkét egyszer sem.

1.4.7.2 Példaprogram

Az alábbi egy üres veremmel elfogadó veremautomata, mely felismeri a helyes zárójelezések nyelvét.

```
#pragma epushdown
start Z;
term 0, 1;
void main()
{
    input(0 1 0 0 1 0 1 1);
    q0: state
    {
        0: { Z = 0 Z | 0 = 0 0; q0; }
        1: { 0 = eps; q0; }
        eps: { Z = eps; q0; }
    }
}
```

Állapotok halmaza:	$\{q_0\}$	
Kezdőállapot:	q_0	
Átmenetek:	$\delta(q_0, 0, Z) = (q_0, 0Z),$ $\delta(q_0, 1, 0) = (q_0, \varepsilon)$	$\delta(q_0, 0, 0) = (q_0, 00),$ $\delta(q_0, \varepsilon, Z) = (q_0, \varepsilon)$

2 Irodalomjegyzék

1. Minimalist GNU for Windows. [Online] <http://www.mingw.org/>.
2. **Dr. Hunyadvári, László.** Automaták és formális nyelvek II. [Online] <http://aszt.inf.elte.hu/~hunlaci/fnyau2.pdf>.
3. *Delphi Yacc & Lex.* [Online] <http://www.grendelproject.nl/dyacclex/>.
4. TNT Unicode Controls. [Online] <http://www.yunqa.de/delphi/doku.php/products/tntunicodecontrols/index>.
5. **Swett, Justin.** A Heap ADT (Priority Queue) Example in Delphi. [Online] <http://cc.embarcadero.com/item/17246>.