Eris Esgf Compiler

1 Úvod

2 Gramatika v ESGF

2.1 Definice

2.1.1 Jména gramatik a jména balíků

2.1.2 Jména pravidel

2.1.3 Symboly

2.1.4 Komentáře

2.2 Hlavička gramatiky

2.2.1 Identifikující hlavička

2.2.2 Jméno gramatiky

2.2.3 Import

2.3 Tělo gramatiky

2.3.1 Definice pravidla

2.3.2 Rozšíření pravidla

2.3.3 Užití speciálních jmen pravidel

2.4 Příklady

2.4.1 Příklad 1

2.4.2 Příklad 2

2.4.3 Příklad 3

2.4.4 Příklad 4

2.4.5 Příklad 5 Vkládání gramatik

2.4.6 Příklad 6

2.4.7 Příklad 7

3 Gramatika v ELF

3.1 Příklady

3.1.1 Příklad 1

3.1.2 Příklad 2

3.1.3 Příklad 3

3.1.5 Příklad 5 Vkládání gramatik

3.1.6 Příklad 6

3.1.7 Příklad 7

4 Literatura

1 Úvod

ESGF compiler je program, který převede gramatiku zapsanou v Eris Speech Grammar Format (ESGF) do Eris Lattice Format (ELF). Syntaxe ESGF vychází z Java Speech Grammar Format (JSGF V1.0), ELF vychází z HTK-SLF rozšířené o dynamické vkládání gramatik a jiné drobnosti. Zde si zavedeme omezení, že nebudeme užívat Unicode znakovou sadu, i když do budoucna možná bude potřeba na Unicode přejít. Pokud se budeme někde zmiňovat o Unicode znacích, tak si to přeložíme jako ANSI znaky.

2 Gramatika v ESGF

Gramatika je reprezentována pravidly, která jsou popsána pomocí ESGF. Jedná se o bezkontextovou gramatiku, kterou je ale možno přepsat na regulární gramatiku, protože předpokládáme věty konečné délky.

Nyní si popíšeme základní pojmenování a strukturální mechanismy. Následně základní části gramatiky - hlavičku gramatiky a tělo gramatiky. Hlavička gramatiky definuje jméno gramatiky a seznam importovaných pravidel a gramatik. Tělo gramatiky definuje pravidla dané gramatiky jako kombinaci možného řečeného textu a dalších pravidel. Nakonec si ukážeme příklady gramatik zapsaných v ESGF.

2.1 Definice

2.1.1 Jména gramatik a jména balíků

Každá gramatika definovaná podle ESGF má jednotné jméno, které je definováno v hlavičce gramatiky. Možné struktury jmen gramatik jsou:

```
jménoBalíku.jednoduchéJménoGramatiky
jménoGramatiky
```

První tvar (jméno balíku + jednoduché jméno gramatiky) se nazývá *plné jméno gramatiky.* Druhý tvar se nazývá *jednoduché jméno gramatiky.* Příklady jmen gramatik:

```
com.sun.speech.apps.numbers
edu.unsw.med.people
examples
```

Jméno balíku a gramatiky má stejný formát jako balíky a třídy v programovacím jazyku Java. Plné jméno gramatiky je tečkami oddělený seznam *Java identifikátorů* *(pozn.1)*.

Konvence pojmenování gramatik je stejná jako u tříd v programovacím jazyku Java. Konvence minimalizuje možnost kolize jmen. Jméno balíku by mělo být:

```
obrácenéJménoDomény.jménoBalíku
```

Například pro *com.sun.speech.apps.numbers*, je *com.sun* obrácená internetová doména Sun, *speech.apps* je

jméno lokálního balíku a *numbers* je jméno gramatiky (soubor gramatiky má stejné jméno jako gramatika sama).

Pozn. 1
Java identifikátor je délkou neomezená posloupnost Unicode znaků. První znak je písmeno nebo znak z množiny speciálních symbolů včetně '$' a '_'. Následující znaky obsahují písmena, číslice, speciální symboly a další znaky.

2.1.2 Jména pravidel

Gramatika se skládá z množiny pravidel, která společně definují co může být řečeno. Pravidla jsou kombinací textů, které mohou být vysloveny, a odkazů na jiná pravidla. Každé pravidlo má jedinečné jméno. Odkaz na pravidlo je reprezentován pomocí jména pravidla vloženého do závorek < > (menší - větší).

Platné jméno pravidla se podobá Java identifikátoru, ale povoluje ještě přídavné symboly. Platné jméno pravidla má neomezenou délku posloupnosti Unicode znaků (pozn.2) s následujícím omezením:

Platné jméno pravidla nesmí obsahovat znaky: *; = / ( ) [ ] { } < > "*

Tvůrci gramatik si musí být vědomi následujících omezení:

1. Jména pravidel jsou přesně porovnávána. Jsou tedy závislá na velikosti písmen. Například <*Name* >, <*NAME*> a <*name*> jsou rozdílná jména pravidel.

2. Ve jménech pravidel nejsou povoleny bílé znaky.

Jména pravidel <*NULL*> a <*VOID*> jsou rezervována. Také jména pravidel (pozn.3) <*BEGIN_GRAMMAR*>, <*END_GRAMMAR*>, <*EXTERN_IN_GRAMMAR*>, <*EXTERN_OUT_GRAMMAR*>, <*VIRTUAL*>, <*MUMBLE*> a <*SIL*> jsou rezervována. Tato speciální pravidla budou probrána v dalších částech.

Unicode znaková sada obsahuje většinu znaků živých jazyků, takže gramatika může být napsána v Čínštině, Japonštině atd.

Kvalifikovaná a plně kvalifikovaná jména

Ačkoli jména pravidel jsou jednotná v rámci jedné gramatiky, oddělené gramatiky mohou použít stejné jednoduché jméno gramatiky. V další kapitole si ukážeme výraz *import*, který nám umožní přistupovat z jedné gramatiky do druhé pro požadovaná pravidla. Když dvě stejné gramatiky používají stejné jméno pravidla, odkaz na jméno pravidla může být dvojsmyslný. *Kvalifikovaná jména* a *plně kvalifikovaná jména* jsou používána pro odkazy mezi gramatikami, aby nedocházelo k dvojsmyslnosti.

Plně kvalifikované jméno obsahuje *plné jméno gramatiky* a *jednoduché jméno pravidla*:

```
<com.sun.greetings.english.hello>
<com.sun.greetings.deutsh.gutenTag>
```

Kvalifikované jméno obsahuje pouze *jednoduché jméno gramatiky* a *jméno pravidla*. Je to užitečná zkrácená forma. Například:

```
<english.hello>
<deutsh.gutenTag>
```

Následující podmínky souvisí s užíváním jmen pravidel:

- Kvalifikované jméno a plně kvalifikované jméno pravidla nesmí být užito na levé straně definice pravidla.
- Výraz import musí užít plně kvalifikovaného jména pravidla.
- Na lokální pravidla může být odkazováno kvalifikovaně nebo plně kvalifikovaně s užitím následující formy: < *lokálníJménoGramatiky.jménoPravidla* >.

Vyhodnocování pravidel

Za chybu se považuje užití dvojsmyslného jména pravidla. Následující definice upravují postup při vyhodnocování pravidel:

- Lokální pravidla mají přednost. Jestliže lokální pravidlo a jedno nebo více importovaných pravidel mají stejné jméno <*jménoPravidla*>, jednoduché jméno pravidla <*jménoPravidla*> je odkaz na lokální pravidlo.
- Jestliže dvě nebo více importovaných pravidel mají stejné jméno <*jménoPravidla*>, ale není definováno lokální pravidlo stejného jména, potom odkaz užívající jednoduché jméno pravidla je dvojsmyslný a způsobí chybu. Aby bylo možné pravidla vyhodnotit, musí být odkazovány pomocí kvalifikovaných nebo plně kvalifikovaných jmen.
- Jestliže dvě nebo více importovaných pravidel mají stejné jméno a pochází z gramatik se stejným jednoduchým jménem (ale nutně z rozdílných balíků), potom odkaz užívající jednoduché jméno pravidla nebo kvalifikované jméno pravidla je dvojsmyslný a způsobí chybu. Proto musí být pravidla odkazována pomocí plně kvalifikovaných jmen.
- Odkaz pomocí plně kvalifikovaného jména není nikdy dvojsmyslný. Když nemůže být jméno pravidla vyhodnoceno (není definované lokálně a není definované ani jako importované veřejné pravidlo), tak záleží na rozpoznávacím softwaru jak se zachová (pozn.4).

Speciální jména pravidel

ESGF definuje tato speciální pravidla: <*NULL*>, <*VOID*>, <*BEGIN_GRAMMAR*>, <*END_GRAMMAR*>, <*EXTERN_IN_GRAMMAR*>, <*EXTERN_OUT_GRAMMAR*>, <*VIRTUAL*>, <*MUMBLE*> a <*SIL*>. Tato pravidla jsou univerzálně definovaná, jsou přístupná aniž by byla explicitně importována nějaká gramatika a nemohou být předefinována. Všechna jména jsou plně kvalifikovaná, takže není potřeba žádné další kvalifikace.

<*NULL*> definuje takové pravidlo, které vždy zajistí, aby daná větev byla platná - to znamená, že je platná aniž by uživatel řekl jediné slovo.

<*VOID*> definuje takové pravidlo, které nemůže být nikdy vysloveno. Vložení <*VOID*> do posloupnosti slov automaticky zapříčiní, že daná posloupnost slov nemůže být vyslovena.

<*BEGIN_GRAMMAR*> definuje takové pravidlo, které označí vstupní místo do gramatiky.

<*END_GRAMMAR*> definuje takové pravidlo, které označí výstupní místo z gramatiky.

<*EXTERN_IN_GRAMMAR*> definuje takové pravidlo, které označí místo pro vstup z jiné gramatiky.

<*EXTERN_OUT_GRAMMAR*> definuje takové pravidlo, které označí místo pro výstup do jiné gramatiky.

<*VIRTUAL*> definuje takové pravidlo, které zajistí na daném místě vložení *virtual* uzlu.

<*MUMBLE*> definuje takové pravidlo, které zajistí na daném místě vytvoření *mumble* modelu.

<*SIL*> definuje takové pravidlo, které zajistí na daném místě vytvoření *sil* modelu.

Pravidla <*NULL*> a <*VOID*> se užívají většinou při speciálních okolnostech. Mohou být použity k zablokování, případně aktivování jednotlivých částí gramatiky nebo ke kontrole rekurze.

2.1.3 Symboly

Symboly, často nazývané *terminální symboly*, jsou částí gramatiky, která definuje co může uživatel říci. Velice často je symbol ekvivalentní slovu. Symboly v ISOLATIN mohou vypadat následovně:

```
ahoj
ostrov
```

nebo posloupnost symbolů oddělených mezerami:

```
toto je test
otevři složku
```

V ESGF je symbol posloupnost znaků ohraničená mezerami, uvozovkami nebo rozdělená pomocí dalších symbolů, které mají zvláštní význam v gramatice:

```
; = | * + / < > ( ) [ ] { } /* */ //
```

Symbol je odkaz na slovo ve *slovníku rozpoznávacího softwaru*. Slovník rozpoznávacího softwaru určuje výslovnost daného slova. Ve speciálních případech může tvůrce gramatiky pomocí mechanismu popsaného v dalších částech přímo definovat *výslovnost* daného slova. Na základě výslovností je rozpoznávací slovník schopen rozpoznávat promluvy.

ESGF umožňuje *vícejazyčné* gramatiky, které obsahují symboly z více než jednoho jazyka. Avšak většina rozpoznávačů pracuje pouze s jedním jazykem, takže typická gramatika obsahuje pouze jeden jazyk. Je pouze na aplikaci, aby nahrála gramatiku do rozpoznávače a zajistila odpovídající jazykovou podporu. Jako příklad může sloužit následující vícejazyčné pravidlo:

```
<no> = no | nein | nao | non | nem;
```

Většina rozpoznávačů má vyčerpávající slovník pro každý jazyk, který podporuje. Přesto není nikdy možné, aby obsahovaly 100% jazyka. Například jména, technická slova a cizí slova nejsou často ve slovníku. Jsou zde čtyři možnosti jak naložit s chybějícími symboly:

- Aplikace nebo uživatel může přidat symbol a jeho výslovnost do slovníku rozpoznávače, aby bylo zajištěno správné rozpoznávání.
- Dobré rozpoznávače jsou schopny odhadnout výslovnost mnoha slov, která nejsou obsažena ve slovníku.
- Tvůrce gramatiky může přímo definovat výslovnost symbolu, která je posléze předána rozpoznávači.
- Jestliže žádný z předchozích bodů není aplikován, pak je chování určeno rozpoznávačem. Ve většině případů bude nenalezený symbol označen jako nepromluvitelný (ekvivalentní <*VOID*>), nebo způsobí chybu nebo výjimku. Například v Java Speech API jsou nedefinované symboly označeny jako nepromluvitelné.

Symboly nemusí být pouze normální psaná slova jazyka, ale předpokládá se, že symboly budou přesně definovány ve slovníku rozpoznávače, popřípadě jejich výslovnost přímo definována při tvorbě gramatiky. Například abychom mohli zachytit rozdílné výslovnosti slov s vlivem na kontext.

Symboly v uvozovkách

Symbol nemusí být slovo. Symbol může být posloupnost slov nebo znak. Uvozovky mohou být užity k ohraničení několikaslovního symbolu a speciálních znaků. Například:

```
metro v "Dlouhé Loučce"
"+"
```

Víceslovní symboly jsou užitečné, když se výslovnost slov mění v závislosti na kontextu. Víceslovní symboly mohou také zjednodušit zpracování výsledků. Například získáme jeden symbol pro "Dlouhá Loučka", "Újezd u Uničova" atd.

Symboly v uvozovkách mohou být ve slovníku rozpoznávače stejně jako jiné symboly. Jestliže víceslovný symbol v uvozovkách nebyl nalezen ve slovníku, pak prvotní chování je určit výslovnost každého mezerou

odděleného slova uvnitř uvozovek.

Abychom mohli uvést uvozovku v symbolu, tak jí musí předcházet obrácené lomítko '\'. Podobně jestliže chceme, aby uvnitř symbolu bylo obrácené lomítko, tak mu musí předcházet další obrácené lomítko. Totéž platí i pro složené závorky '{}' používané v definici unárního operátoru tag popisovaném v dalších částech.
Například:

\\ \" \{ \}

Mezery uvnitř uvozovek jsou významné.

Znaky a interpunkční znaménka

Většina rozpoznávačů má schopnost přeložit většinu znaků a interpunkčních znamének. Například v případě angličtiny apostrofy (Mary's, it's).
Přesto je mnoho textů, které je těžké převést jednoznačně. Proto by tvůrce gramatiky měl použít symboly tak, aby se co nejvíce podobaly tomu, jak se budou vyslovovat, popřípadě jejich výslovnost přímo definovat.
Například následující:

- Čísla: "0 1 2" by měla být rozepsána na "nula jedna dva". Podobně "volej 555 1234" by mělo být rozepsáno na "volej pět pět pět jedna dva tři čtyři".
- Datum: "Dec 25, 1999" by mělo být psáno jako "prosinec dvacátého pátého tisíc devět set devadesát devět".
- Zkratky a akronymy: "p." by mělo být psáno jako "pan" atd.
- Speciální znaky: '&' jako "ampersand" nebo "a", "+" jako plus atd.

2.1.4 Komentáře

Komentáře mohou být jak v hlavičce tak v těle. Styl komentářů odpovídá jazyku C++.

- */* test *\/* Všechny znaky mezi /* a */ jsou ignorovány.
- *// text* Všechny znaky od // až do konce řádku jsou ignorovány.

Vkládání komentářů do komentářů nemá žádný speciální význam. To znamená, že se s vnořenými komentáři pracuje jako s běžnými texty.

Komentáře se mohou vyskytovat všude v definici gramatiky, kromě symbolů v uvozovkách a tagů.

Pozn. 2
Unicode znaky odpovídají následujícím pravidlům:

- Znaky jsou písmena, číslice, speciální symboly a další symboly.
- Znaky jsou systém rozdělovacích znamének: *+ - : ; , = | / \ ( ) [ ] @ # % ! ^ & ~*

Pozn. 3
Jména pravidel *<BEGIN_GRAMMAR>, <END_GRAMMAR>, <EXTERN_IN_GRAMMAR>, <EXTERN_OUT_GRAMMAR>, <VIRTUAL>, <MUMBLE>* a *<SIL>* jsou rozšířením ESGF oproti JSGF. Proto jsme zavedli ESGF.

Pozn. 4
Doporučuje se vypsat chybu o neexistujícím pravidle.

2.2 Hlavička gramatiky

Jeden soubor definuje jednu gramatiku. Definice gramatiky obsahuje dvě části: *hlavičku gramatiky* a *tělo gramatiky*. Hlavička gramatiky zahrnuje identifikující hlavičku, definuje jméno gramatiky a uvádí, která pravidla se mají importovat z ostatních gramatik. Tělo definuje pravidla gramatiky, z nichž některá mohou být veřejná.

2.2.1 Identifikující hlavička

ESGF soubor začíná identifikující hlavičkou. Tato hlavička informuje o tom, že soubor obsahuje ESGF a určuje v jaké verzi ESGF byl napsán (nyní V1.0). Dále hlavička volitelně specifikuje znakovou sadu, ve které byl soubor napsán, případně ještě jazyk. Identifikující hlavička je ukončena středníkem a začíná se na nové řádce.

Formát hlavičky je následující:

```
#ESGF verze znaková-sada jazyk;
```

Příklady hlaviček:

```
#ESGF V1.0;
#ESGF V1.0 ISO8859-5;
#ESGF V1.0 JIS ja;
```

První příklad neuvádí typ znakové sady ani jazyk, takže se bude uvažovat počáteční nastavení. V US počáteční nastavení může být ISO8859-1 (standardní znaková sada) a "en" (symbol pro Angličtinu). Druhý příklad definuje ISO8859-5 znakovou sadu (Cyrilic), ale je předpokládán počáteční jazyk. Poslední příklad definuje užití JIS (jedna z japonských znakových sad) a definuje jazyk "ja" (Japonština).

Java platforma umí zpracovat mnoho znakových sad, které interně převede do Unicode. Jestliže se užije Unicode, potom je ESGF připraveno pro zápis gramatik ve skoro všech živých jazycích.

Znak *'#'* musí být prvním znakem v ESGF souboru a všechny znaky v identifikující hlavičce musí být ASCII podmnožinou užité znakové sady.

2.2.2 Jméno gramatiky

Jméno gramatiky musí být definováno jako první výraz gramatiky. Gramatika musí užít plného jména gramatiky (plné jméno + jednoduché jméno gramatiky). Proto formát definice jména gramatiky musí vypadat následovně (můžeme si vybrat ze dvou tvarů):

```
grammar jménoBalíku.jednoduchéJménoGramatiky;
grammar jménoGramatiky;
```

Například:

```
grammar com.sun.speech.apps.numbers;
grammar edu.unsw.med.people;
grammar examples;
```

2.2.3 Import

Hlavička gramatiky také může volitelně obsahovat deklaraci *import. Import* deklarace následuje za deklarací *grammar* a musí být před tělem gramatiky (definicí pravidel). *Import* deklarace povoluje jedno nebo všechna veřejná pravidla z importované gramatiky. Formát deklarace výrazu *import* může být jeden ze dvou následujících:

```
import <plněKvalifikovanéJménoPravidla>
import <plnéJménoGramatiky.*>
```

Například:

```
import <com.sun.speech.apps.index.1stTo31st>
import <com.sun.speech.apps.numbers.*>
```

První příklad importuje jedno plně kvalifikované pravidlo <1stTo31st> z gramatiky <com.sun.speech.apps.index>. Importované pravidlo <1stTo31st> musí být veřejné v importované gramatice.

Užití hvězdičky v druhém příkladě způsobí importování všech veřejných pravidel z importované gramatiky. Jestliže gramatika definuje tři veřejná pravidla *<digits>, <teens> a <zeroToMilion>*, potom všechna tři mohou být odkazována lokálně.

Připomínáme, že jméno gramatiky a jméno pravidla nebo hvězdička je vyžadována, a proto není možné výraz *import* zapsat následovně:

```
import <jménoPravidla>
```

Na importované pravidlo může být odkazováno třemi různými způsoby: pomocí jednoduchého jména (např. *<digits>*), pomocí kvalifikovaného jména (např. *<numbers.digits>*), nebo pomocí plně kvalifikovaného jména (např. *<com.sun.speech.apps.numbers.digits>*).

Dále připomínáme, že výraz *import* je volitelný zejména tehdy, jestliže vždy používáme plně kvalifikovaná jména pravidel.

2.3 Tělo gramatiky

2.3.1 Definice pravidla

Gramatika definuje *pravidla*. Každé pravidlo je definováno pomocí *definice pravidla*. Pravidlo může být v gramatice definováno pouze jedinkrát. Pořadí definice pravidel není významné.

Uveďme si pět možných vzorů definice pravidla:

```
<jménoPravidla> = rozšířeníPravidla;
public <jménoPravidla> = rozšířeníPravidla;
textree <jménoPravidla> = rozšířeníPravidla;
public lextree <jménoPravidla> = rozšířeníPravidla;
lextree public <jménoPravidla> = rozšířeníPravidla;
```

Definice pravidla má následující složky:

- volitelnou *public* deklaraci,
- volitelnou *lextree* deklaraci,
- jméno pravidla, které definujeme,
- znak rovnosti '=',
- rozšíření pravidla,
- znak ukončení (středník ';').

Mezery jsou přeskočeny před definicí, mezi klíčovými slovy *public* a *lextree* a jménem pravidla, okolo znaku rovnosti a okolo středníku. Mezery jsou významné uvnitř rozšíření pravidla.

Rozšíření pravidla definuje jak může být pravidlo vysloveno. Logicky to bývá kombinace *symbolů* (test, který může být vysloven) a *odkazů* na jiná pravidla. Termín "rozšíření" je použit, protože definuje jak má být pravidlo rozšířeno, když je vysloveno. Jedno pravidlo může být rozšířeno na mnoho slov a pravidel, která mohou být sama o sobě rozšířena. V dalších částech si uvedeme možná rozšíření.

Veřejná pravidla

Jakékoli pravidlo v gramatice může být definováno jako veřejné pomocí klíčového slova *public*. Veřejná pravidla mají tři možná užití:

- Může být na ně odkazováno z jiné gramatiky pomocí plně kvalifikovaného jména nebo pomocí deklarace *import* a jednoznačného odkazu.
- Mohou být užita jako *aktivní* pravidla pro rozpoznávání. To znamená, že mohou být užita rozpoznávačem k určení co bylo promluveno řečníkem.
- Může být na ně odkazováno lokálně - chovají se tedy jako neveřejné uvnitř gramatiky.

Bez deklarace *public* je pravidlo implicitně *private* (soukromé) a může být na něj odkazováno jen uvnitř gramatiky, kde bylo definováno.

Pravidla lexikálních stromů

Jakékoli pravidlo v gramatice může být definováno jako lexikální strom pomocí klíčového slova *lextree*. Takové pravidlo bude přeloženo jako lexikální strom, který bude na výstupu compileru v samostatném setu. Tento postup umožní na lexikální strom použít odlišný optimalizovaný dekodér. Na místech odkazů na pravidlo lexikálního stromu se zajistí externí reference a na toto pravidlo je tedy možno se odkazovat stejně jako na jakékoli jiné pravidlo.

Rozšíření pravidla lexikálního stromu je odlišné od rozšíření ostatních pravidel. Do lexikálního stromu mohou být zařazeny jen jednotlivá slova popřípadě kombinace slov. Rozšířením pravidla lexikálního stromu tedy mohou být pouze alternativy posloupností symbolů (viz další kapitola).

Příklad definice pravidla lexikálního stromu:

```
    lextree = Praha | Prachatice | Prachařov | Kladno | Kladruby;
```

2.3.2 Rozšíření pravidla

Nejednodušší rozšíření je odkaz na symbol nebo odkaz na pravidlo. Například:

```
<a> = slon;
<b> = <x>
<c> = <com.acme.grammar.y>
```

Pravidlo <*a*> se jednoduše rozšíří na symbol "slon". Proto, aby bylo vysloveno <a>, musí být promluveno slovo "slon".

Pravidlo <*b*> se rozšíří na <*x*>. To znamená, že aby bylo vysloveno <*b*>, musí být promluveno něco, co odpovídá <*x*>. Podobně, aby bylo vysloveno <*c*>, musí být promluveno něco, co odpovídá pravidlu <*com.acme.grammar.y*>.

Formálně řečeno, následující rozšíření jsou povolená:

- Libovolný symbol.
- Odkaz na libovolné veřejné nebo neveřejné pravidlo definované uvnitř té samé gramatiky.
- Odkaz na libovolné veřejné pravidlo definované v jiné gramatice, která byla importována.
- Odkaz na veřejné pravidlo definované v jiné gramatice, pokud je odkaz plně kvalifikované jméno pravidla.

Tento postup je uplatňován pouze lokálně. To znamená, že lze odkazovat z gramatiky *grammar1* pomocí pravidla do jiné gramatiky *grammar2* na veřejné pravidlo, a to může odkazovat na interní (v gramatice *grammar2*) neveřejné pravidlo. Jinak řečeno, definice pravidla může nepřímo odkazovat na soukromé pravidlo jiné gramatiky přes veřejné pravidlo téže gramatiky.

Prázdná definice není přípustná.

```
<d> = ; // není možné
```

Definice pravidla zastupujících <NULL> nebo <VOID> jsou možné.

```
<e> = <NULL> // je přípustné
<f> = <VOID> // je přípustné
```

Následující část vysvětlí, jak mnohem složitější pravidla mohou být tvořena pomocí logických kombinací přípustných rozšíření:

- *Skládání* posloupností rozšíření a množin alternativních rozšíření.
- *Seskupování* užívající kulaté a hranaté závorky.
- *Unární operátory* zajišťující opakování rozšíření.
- Připojení na aplikaci závislých *tagů* k rozšíření.

Posloupnosti

Pravidlo smí být definováno jako posloupnost rozšíření. Posloupnost přípustných rozšíření, každé oddělené mezerou, je také přípustné rozšíření. Například jelikož všechny symboly jsou přípustné rozšíření, proto následující pravidla jsou přípustná:

```
<kde> = bydlím v Dlouhé Loučce;
<prohlášení> = tento <objekt> je <OK>
```

Chceme-li vyslovit danou posloupnost, pak to musíme udělat v daném pořadí. Pokud chceme vyslovit pravidlo <*prohlášení*>, pak musíme říci místo pravidel <*objekt*> a <*OK*> jim odpovídající symboly.

Prvky posloupnosti mohou být libovolná přípustná rozšíření včetně těch, která si ještě vysvětlíme.

Alternativy

Pravidlo smí být definováno jako *množina alternativních* rozšíření oddělených znakem '|'. Například:

```
<jména> = Michal | Filip | Luboš | Martin | <dalšíJména>
```

K vyslovení pravidla <*jména*> musí řečník vyslovit právě jedno jméno z množiny alternativ. Řečník může říci "Michal", "Filip" atd., ale nesmí říci "Luboš Martin".

Posloupnosti mají vyšší prioritu než alternativy. Například

```
<státy> = Velká Morava | Česká Republika| Jihoafrická Republika;
```

je množina tří alternativ, kde je každá jménem státu.

Prázdná alternativa není přípustná.

```
<jména> = Michal | | Michala; // není přípustné
<jména> = Michal | Michala | ; // není přípustné
```

Váhy

Ne všechny cesty v gramatice jsou stejně pravděpodobné. Váhy mohou být připojeny ke každému prvku množiny alternativ. Znamenají pravděpodobnost každé alternativy, že bude vyslovena. Váha je nezáporné reálné číslo uvnitř dvou lomítek *'/'*, například */3.14/*. Čím je váha větší, tím je více pravděpodobné, že bude daná alternativa vyslovena. Například:

```
<barva> = /10/ červená | /2/ černá | /1/ zelená;
<velikost> = /0.5/ malá | /0.1/ střední | /0.2/ velká;
```

Váhy by měly odpovídat pravděpodobnosti výskytu v promluvách. V prvním příkladě je "červená" 10 krát pravděpodobnější než "zelená" a 5 krát pravděpodobnější než "černá".

Váhy musí splňovat následující pravidla:

- Jestliže je váha uvedena pro jednu alternativu, pak musí být uvedeny váhy pro ostatní alternativy z množiny alternativ.
- Váhy jsou reálná čísla.
- Jedině reálné číslo a mezery jsou povoleny uvnitř lomítek.
- Váha musí být nezáporná. Nulová váha nám říká, že daná alternativa nebude nikdy vyslovena - je to ekvivalentní <*VOID*>. (Nulové váhy mohou být užitečné při vývoji gramatiky.)
- Alespoň jedna váha musí být pozitivní.

Odvození patřičných vah je obtížné a odhadnutí ne vždy zlepšuje přesnost rozpoznávání. Správné váhy jsou většinou získány pečlivým studiem řeči a textových dat.

Seskupování

Přípustné rozšíření může být seskupeno pomocí kulatých závorek *'()'*. Seskupování má vysokou prioritu, a proto může být užito k zajištění správné interpretace pravidel. Je také užitečné při zvyšování čitelnosti gramatiky. Například, protože posloupnosti mají větší prioritu než alternativy, kulaté závorky jsou nutné k definování následujícího pravidla: "prosím otevřete" a "prosím zavřete":

```
<akce> = prosím (otevřete | zavřete);
```

Následující příklad ukáže posloupnost tří položek. Položky jsou alternativy, které musí být v závorkách, abychom dosáhli žádané gramatiky:

```
<příkaz> = (otevřete | zavřete)(okna | dveře)(hned | za chvíli);
```

Aby promluva odpovídala pravidlu <*příkaz*>, musí obsahovat vždy jedno slovo z každé množiny alternativ. Například "otevřete dveře hned" nebo "zavřete okna za chvíli".

Je přípustné mít uvnitř závorek jedno rozšíření, avšak není přípustné mít prázdné závorky.

```
( start ) // je přípustné
( <end> ) // je přípustné
( ) // není přípustné
```

Volitelné seskupování

Přípustné rozšíření může být volitelně seskupeno pomocí hranatých závorek *'[]'*. Hranaté závorky mohou být umístěny okolo jakékoli definice pravidla a informují nás, že daný obsah je volitelný. Jinak je ekvivalentní kulatým závorkám a má stejnou prioritu.

Například:

```
<prosba> = prosím | pěkně prosím;
public <příkaz> = [ <prosba> ] nenabourej se;
```

Pravidlo <*příkaz*> umožňuje řečníkovy říci "nenabourej se" a volitelně "prosím nenabourej se" nebo"pěkně prosím nenabourej se".

Prázdné závorky jsou nepřípustné.

```
[ ] // není přípustné
```

Unární operátory

V ESGF jsou tři *unární operátory*. Jsou to: hvězdička, plus a tag. Operátory mají následující společné vlastnosti:

- Unární operátor může být připojen k přípustnému rozšíření.

- Mají vysokou prioritu: přísluší k bezprostředně předcházejícímu rozšíření pravidla.
- Pouze jeden unární operátor může být připojen k rozšíření pravidla (pro operátor tag platí výjimka).
- Mezery mezi rozšířením pravidla a připojeným operátorem jsou přeskočeny.

Jelikož priorita unárních operátorů je vyšší než posloupností nebo alternativ, kulaté závorky jsou potřebné k seskupení posloupnosti případně alternativ, abychom mohli užít operátor na celou skupinu.

Operátor hvězdička

Rozšíření pravidla následované znakem hvězdička *'*'* znamená, že rozšíření nemusí být řečeno nebo může být řečeno jednou nebo vícekrát. Například:

```
<příkaz> = <prosba>* nenabourej se;
```

umožňuje řečníkovi říci "prosím nenabourej se", "prosím pěkně prosím nenabourej se" nebo umožní ignorovat pravidlo <*prosba*>.

Jako unární operátor má vysokou prioritu. Proto bude následující příklad:

```
<písnička> = zpívej Dlouhá Loučka *;
```

interpretován tak, že se bude opakovat pouze "Loučka" (např. "zpívej Dlouhá Loučka Loučka Loučka"). Dá se tedy interpretovat následovně: "zpívej Dlouhá", "zpívej Dlouhá Loučka" atd., ale nemůžeme zpívat "zpívej Dlouhá Loučka Dlouhá Loučka". Uvozovky nebo závorky mohou upravit oblast působnosti operátoru hvězdička. Například:

```
<písnička> = zpívej (Dlouhá Loučka)*;
```

odpovídá již zmíněnému "zpívej Dlouhá Loučka Dlouhá Loučka".

Operátor plus

Rozšíření pravidla následované znakem plus *'+'* znamená, že rozšíření může být řečeno jednou nebo vícekrát. Například:

```
<příkaz> = + nenabourej se;
```

umožňuje řečníkovi říci "prosím nenabourej se", "prosím pěkně prosím nenabourej se" atd. Ale není přípustná promluva "nenabourej se".

Tag

Tagy poskytují vývojářům gramatik připojit aplikačně závislá data k částem definic pravidel. Aplikace často užije tagy ke zjednodušení nebo ke zkvalitnění zpracování výsledků rozpoznávání.

Kromě speciálních tagů jsou tyto rozpoznávačem připojeny k rozpoznané promluvě a předány aplikaci. Rozpoznávač určuje jak jsou tagy předávány.

Tag je unární operátor. Jako takový může být připojen k jakémukoli přípustnému rozšíření pravidla. Tag je řetězec ohraničený složenými závorkami *'{}'*. Všechny znaky včetně mezer jsou považovány za součást tagu. Prázdné složené závorky jsou přípustné a tag bude vyhodnocen jako řetězec nulové délky.

Tag se připojí k bezprostředně předcházejícímu rozšíření pravidla (případné mezery jsou ignorovány). Například:

```
<akce>= prosím (otevřete {OPEN} | zavřete {CLOSE} );
<státy> = Velká Morava {VM} | Česká Republika {CZ} | Jihoafrická Republika {JAR};
```

Vzhledem k tomu, že tag je unárním operátorem, tak má větší prioritu než posloupnosti nebo alternativy. Například:

```
<objekt>= kniha | časopis | noviny {věc};
```

tag "věc" je připojen pouze k symbolu "noviny". Jestliže použijeme kulaté závorky, mohou být užity tak, aby ke všem symbolům byl přiřazen stejný tag:

```
<objekt> = ( kniha | časopis | noviny ) {věc};
```

Narozdíl od unárních operátorů, může být přiřazeno více tagů k jednomu rozšíření pravidla. Například:

```
<objekt> = kniha {tag1} {tag2} {tag3}; // je přípustné
```

Toto pravidlo je zpracováno jako by bylo v následujícím tvaru:

```
<objekt> = ((kniha {tag1}) {tag2}) {tag3}; // je přípustné
```

Tato možnost je povolena jen u tagů. Následující příklady jsou nepřípustné:

```
<objekt> = kniha * {tag1}; // není přípustné
<objekt> = kniha {tag1} +; // není přípustné
```

Speciální tagy

V ESGF jsou navíc definovány tři speciální tagy, které nejsou přímo předávány dále aplikaci. Tyto tagy slouží k doplňujícímu popisu výslovnosti symbolů, k pojmenování určitých uzlů výsledné sítě a definici pravděpodobnosti zpětného přechodu u *mumble* modelu, a jsou tedy využity přímo při kompilaci ESGF.

Tag ve tvaru *{pt=...}* definuje fonetickou transkripci (tedy přepis výslovnosti) symbolu, ke kterému se váže. Z tohoto důvodu není možno tento tag umístit za speciální pravidla *<NULL>, <VOID>, <BEGIN_GRAMMAR>, <END_GRAMMAR>, <EXTERN_IN_GRAMMAR>, <EXTERN_OUT_GRAMMAR>, <VIRTUAL>, <MUMBLE>* a *<SIL>*. Ve výsledné síti ELF je fonetická transkripce, tedy část tagu za znakem '=', umístěna do jména uzlu, je nastaven příslušný parametr indikující, že jméno uzlu je již foneticky transkribováno, a původní symbol gramatiky je vložen do anotace uzlu. Například:

```
<číslovky> = 1 {pt=jedna} | 2 {pt=dvje} | 3 {pt=tRi} | 4 {pt=CtiRi} | 5 {pt=pjet} | 6 {pt=Sest} | 7 {pt=sedum} | 8 {pt=osum} | 9 {pt=devjet} | 0 {pt=nula};
```

Tag ve tvaru *{nn=...}* definuje jméno uzlu sítě v ELF zavedeného pomocí speciálních pravidel *<BEGIN_GRAMMAR>, <END_GRAMMAR>, <EXTERN_IN_GRAMMAR>, <EXTERN_OUT_GRAMMAR>* a *<VIRTUAL>*. Z tohoto důvodu je tedy tento tag možno umístit pouze za výše uvedená speciální pravidla. Ve výsledné síti ELF je opět část tagu za znakem '=' umístěna do jména uzlu.

```
<výstup> = <EXTERN_OUT_GRAMMAR> {nn=Externí 1};
```

Tag ve tvaru *{mp=...}* definuje pravděpodobnost zpětného přechodu u *mumble* modelu. Z tohoto důvodu je tedy tento tag možno umístit pouze za speciální pravidlo *<MUMBLE>*. Pravděpodobnost, tedy část tagu za znakem '=', musí být reálné číslo z intervalu (0;1).

Tvůrce gramatiky však musí dát pozor zejména na nechtěné vícenásobné využití tagu pro definici fonetické transkripce při použití současně se seskupováním nebo za odkazem na další pravidlo.

Užití tagů

Tagy mohou zjednodušit zpracování výsledků rozpoznávače. Obsah tagu a užití tagu zcela závisí na rozhodnutí vývojáře gramatiky.

Užití tagů může být důležité při zpracování vícejazyčných gramatik. Následuje ukázka čtyř gramatik, každá pro jiný jazyk. Aplikace nahraje gramatiku odpovídající jazyku řečníka do patřičného rozpoznávače. Tagy zajistí stejné výsledky pro všechny jazyky, a tedy zjednoduší aplikaci zpracování výsledků. Typicky jméno gramatiky obsahuje identifikátor jazyka, a tak může být aplikací automaticky nalezen a nahrán.

Pravidlo v češtině:

```
<pozdrav> = (nazdar | dobrá den) {ahoj};
```

Pravidlo v angličtině:

```
<pozdrav> = (howdy | good morning) {ahoj};
```

Pravidlo v japonštině:

```
<pozdrav> = (ohayo | ohayogozaimasu) {ahoj};
```

Pravidlo ve francouzštině:

```
<pozdrav> = (bon jour) {ahoj};
```

Priorita

Následující seznam definuje prioritu v ESGF od nejvyšší k nejnižší:

1. Jméno pravidla uvnitř znaků menší '<' větší '>' a symbol v uvozovkách nebo bez uvozovek.
2. Kulaté závorky '()' na seskupování a hranaté závorky '[]' na volitelné seskupování.
3. Unární operátory ('+', '*' a tag), které jsou aplikovány na bezprostředně předcházející rozšíření pravidla.
4. Posloupnost rozšíření pravidla.
5. Množina alternativ rozdělená znakem '|'.

Rekurze

Rekurze je definování odkazu sám na sebe. Rekurze je silný nástroj, který umožňuje vyjádřit mnoho složitých promluv, které nastávají v mluveném jazyce. Rozpoznávač podporující ESGF povoluje *pravou rekurzi*. V případě pravé rekurze odkazuje pravidlo samo na sebe v poslední části definice.

Například:

```
<příkaz> = <akce> | ( a );
<akce> = zastav | začni | přeruš | ukonči;
```

povoluje následující příkazy: "zastav", "zastav a ukonči", "začni a přeruš a ukonči".

Nepřímá pravá rekurze je také povolena. Nepřímá pravá rekurze je definování odkazu na jiné pravidlo a z tohoto pravidla je definován odkaz na původní pravidlo. Například:

```
<X> = něco | <Y>
<Y> = něco jiného | <X>
```

Nepřímá pravá rekurze se může vyskytovat také mezi gramatikami, ale to se silně nedoporučuje. Tato rekurze může způsobit nepřiměřenou složitost a problémy s úpravou gramatik.

Libovolná pravá rekurze může být přepsána pomocí užití operátoru hvězdička '*' a (nebo) pomocí operátoru plus '+'. Například následující definice pravidla jsou ekvivalentní:

```
<příkaz> = <akce> | (<akce> a <příkaz>);
<příkaz> = <akce> ( a <akce>)*;
```

Ačkoli je možné přepsat pravou rekurzi pomocí operátorů '+' a '*', rekurze je povolena, protože umožňuje některé typy gramatik zapsat jednodušeji a elegantněji. Jiné podoby rekurze (levá rekurze, vložená rekurze) není povolena, protože nelze zajistit přepsání do nerekurzivní podoby.

2.3.3 Užití speciálních jmen pravidel

Užití *<NULL>* a *<VOID>*

Speciální jména *<NULL>* a *<VOID>* mají mnoho užitečných užití.

Odkaz na *<NULL>* jako alternativy je podobný užití volitelného seskupování. Následující definice jsou tedy ekvivalentní:

```
<x> = a | <NULL>
<x> = [ a ];
```

Operátor hvězdička a pravá rekurze mohou být zaměňovány následovně:

```
<x> = <NULL> | a <x>
<x> = a*;
```

Ve dvou předcházejících případech jsou gramatiky identické ve smyslu, že řečník musí říci přesně stejné promluvy. Avšak zde mohou být programové rozdíly v reprezentaci výsledků rozpoznávání poskytnutých rozpoznávačem, když řečník řekl něco přípustného pro gramatiku.

Další užití *<NULL>* a *<VOID>* je v aktivování či deaktivování některých částí gramatiky. Vývojář gramatiky může vytvořit pravidlo *<povolit>* a umístit je před části, které chce povolit či nepovolit. Pak nadefinuje pravidlo *<povolit>* buď prvním nebo druhým způsobem:

```
<povolit> = <NULL>
<povolit> = <VOID>
```

Jestliže je pravidlo *<povolit>* definováno jako *<VOID>*, pak ta část gramatiky, před kterou je uvedeno, je ignorována. Tato vlastnost se hodí při ladění rozsáhlých gramatik.

Užití *<BEGIN_GRAMMAR>* a *<END_GRAMMAR>*

<BEGIN_GRAMMAR> definuje jméno vstupního uzlu sítě pro dynamické vkládání do jiné sítě. V případě nepřítomnosti pravidla ve veřejném pravidle, budou považovány za vstupní všechny první terminální symboly.

Například:

```
<begin> = <BEGIN_GRAMMAR>
```

<END_GRAMMAR> definuje jméno výstupního uzlu sítě pro dynamické vkládání do jiné sítě. V případě nepřítomnosti pravidla ve veřejném pravidle, budou považovány za výstupní všechny poslední terminální symboly.

Například:

```
<end> = <END_GRAMMAR>
```

Užití *<EXTERN_IN_GRAMMAR>* a *<EXTERN_OUT_GRAMMAR>*

<EXTERN_IN_GRAMMAR> definuje takové pravidlo, které označí místo pro vstup z jiné gramatiky.

Například:

```
<extern_in> = <EXTERN_IN_GRAMMAR>
```

<*EXTERN_OUT_GRAMMAR*> definuje takové pravidlo, které označí místo pro výstup do jiné gramatiky.

Například:

```
<extern_out> = <EXTERN_OUT_GRAMMAR>
```

Užití <*VIRTUAL*>

<*VIRTUAL*> definuje takové pravidlo, které zajistí na daném místě vložení *virtual* uzlu. Tento uzel není odstraněn při dalším zpracování a může tedy významně zredukovat počet hran výsledné sítě.

Například:

```
<virtual_uzel> = (mrkev | celer | petržel | česnek | cibule) <VIRTUAL> (jablka |
hrušky | švestky | třešně);
```

Použití pravidla <*VIRTUAL*> i v tomto jednoduchém případu sníží počet hran z dvaceti na devět.

Užití <*MUMBLE*> a <*SIL*>

<*MUMBLE*> definuje takové pravidlo, které zajistí na daném místě vytvoření mumble modelu.

Například:

```
<mumble_promluva> = <MUMBLE> Velká Morava <MUMBLE>
```

<*SIL*> definuje takové pravidlo, které zajistí na daném místě vytvoření sil modelu.

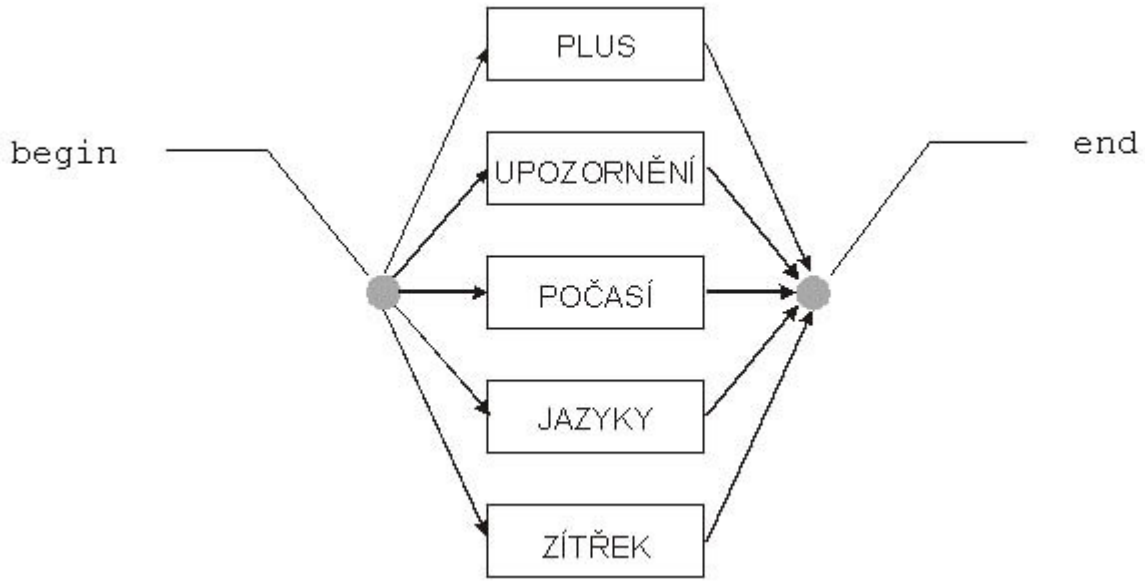
Například:

```
<promluva_se_sil> = Dlouhá Loučka <SIL> to je krásná vesnice;
```

2.4 Příklady

2.4.1 Příklad 1

Chceme následující gramatiku popsat pomocí ESGF:



Podle pravidel ESGF definujeme následující gramatiku v ESGF:

Příklad_1v1.esgf

```
#ESGF V1.0;

grammar Příklad_1v1;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

<words> = plus | upozornění | počasí | jazyky | zítřek;
public <počasí> = <begin> <words> <end>
```

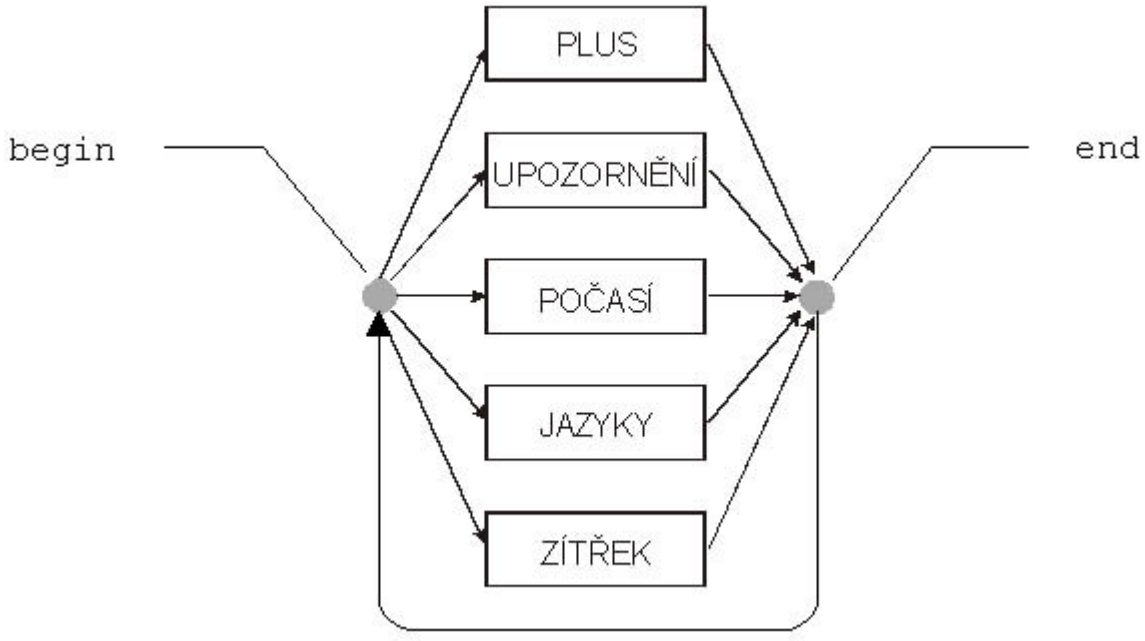
Tuto gramatiku lze vyjádřit také takto:

Příklad_1v2.esgf

```
#ESGF V1.0;

grammar Příklad_1v2;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

public <počasí> = <begin> ( plus | upozornění | počasí | jazyky |
                            zítřek ) <end>
```

2.4.2 Příklad 2

Nyní trochu modifikovaný předchozí příklad zapíšeme opět pomocí ESGF.



Podle pravidel ESGF definujeme následující gramatiku v ESGF:

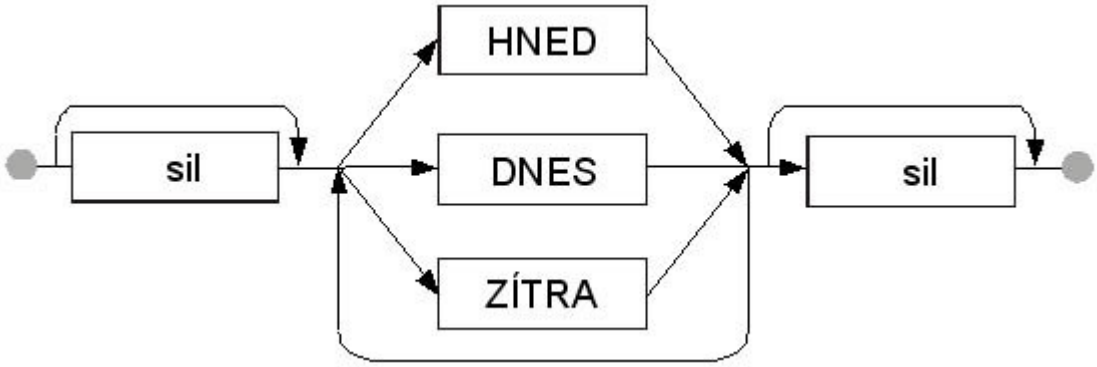
Příklad_2.esgf

```
#ESGF V1.0;

grammar Příklad_2;

<begin> = <BEGIN_GRAMMAR>
```

```
<end> = <END_GRAMMAR>

<words> = plus | upozornění | počasí | jazyky | zítřek;

public <počasí> = <begin> <words>+ <end>
```

2.4.3 Příklad 3

Chtějme následující gramatiku popsat pomocí ESGF:



Podle pravidel ESGF definujeme následující gramatiku v ESGF:

Příklad_3.esgf

```
#ESGF V1.0;

grammar Příklad_3;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

<when> = hned | dnes | zítra ;

public <kdy> = <begin> [ <SIL> ] <when>+ [<SIL> ] <end>
```

2.4.4 Příklad 4

Nyní již bez grafu gramatiky si nadefinujeme malý dialog pro řízení a vytáčení čísla na telefonu.

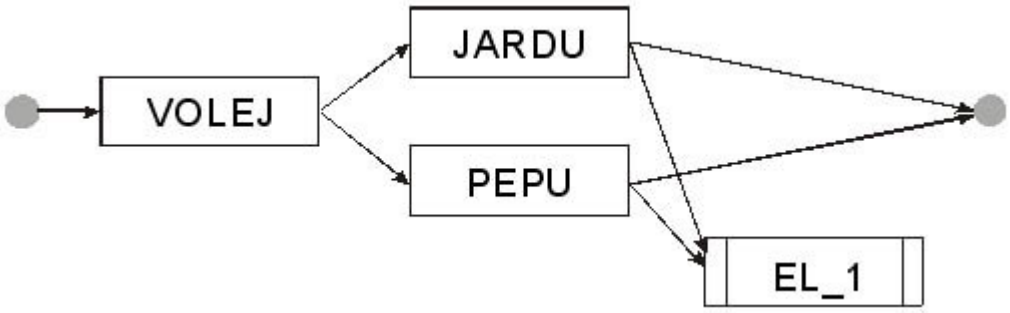
Příklad_4.esgf

```
#ESGF V1.0;

grammar Příklad_4;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

<číslovka> = 1 {pt=jedna} | 2 {pt=dvje} | 3 {pt=tRi} | 4 {pt=CtiRi} |
             5 {pt=pjet} | 6 {pt=Sest} | 7 {pt=sedum} | 8 {pt=osum} |
             9 {pt=devjet} | 0 {pt=nula};
<číslo> = <číslovka> [[pauza] <číslo>];

<zkratka> = zkratka <číslovka> <číslovka>
<zkratku> = zkratku <číslovka> <číslovka>

<telčíslo> = <zkratka> | <číslo>

<příkaz> = vytoč <telčíslo> |
           vlož <zkratku> pro <číslo> |
           znovu vytoč | konec;
```

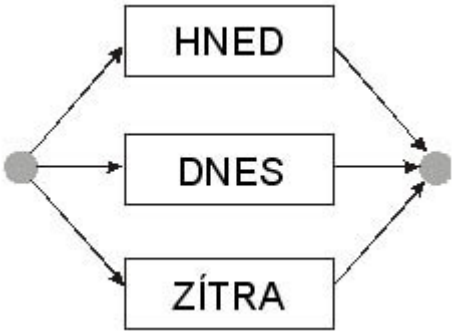
```
<hluk> = mlasknutí | dýchání | pozadí;

public <pokyn> = <begin> ( <příkaz> | <hluk> )+ <end>
```

2.4.5 Příklad 5 Vkládání gramatik

Uvažujme následující gramatiku:



kde na místě EL_1 by měla být následující gramatika:



Tyto dvě gramatiky musíme popsat ESGF formou. Jelikož vyjdeme z předpokladu, že druhá gramatika by se mohla změnit (místo ní použít nějaká jiná), bude nutné popsat tuto gramatikou (celou) pomocí dvou ESGF souborů, kde první gramatika bude exportovat uzel (slovo) tak aby mohlo být nahrazeno později konkrétní gramatikou.

Proto definujeme následující gramatiky v ESGF:

Příklad_5_Jména.esgf

```
#ESGF V1.0;

grammar Příklad_5_Jména;

public <jména> = jardu | pepu;
```

Příklad_5_Hlavní.esgf

```
#ESGF V1.0;

grammar Příklad_5_Hlavní;
import <Příklad_5_Jména.jména>

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>
<EL_1> = <EXTERN_OUT_GRAMMAR> {nn=EL_1};

public <hlavní> = <begin> volej <jména> (<EL_1> | <end>);
```

Pomocí klíčového slova *<EXTERN_OUT_GRAMMAR>* jsme definovali jméno uzlu, který bude využit ke vložení druhé gramatiky.

Příklad_5_Externí.esgf

```
#ESGF V1.0;

grammar Příklad_5_Externí;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

<when> = hned | dnes | zítra ;

public <externí> = <begin> <when> <end>
```

2.4.6 Příklad 6

Nyní opět bez grafu si nadefinujeme gramatiku s užitím mumble a sil modelu.

Příklad_6.esgf

```
#ESGF V1.0;

grammar Příklad_6;

<begin> = <BEGIN_GRAMMAR>
<end> = <END_GRAMMAR>

public <číslovka> = <begin> <MUMBLE> {mp=0.1} Dnes je ale pěkně. <SIL> Že
                    mám pravdu? <MUMBLE> {mp=0.01} <end>
```

2.4.7 Příklad 7

A nyní příklad na lexikální stromy.

Příklad_7.esgf

```
#ESGF V1.0;

grammar Příklad_7;

public lextree <města> = Praha | Prachatice | Prachařov | Kladno |
                         Kladruby | Klatovy;

lextree <jména> = Aleš Pražák | Filip Jurčíček | Luboš Šmídl | Jiří
                  Zahradil | Luděk Müller | Jiří Krůta | Aleš Novosad;

public <veřejné> = pan <jména> cestuje z města <města> do města <města>
```

3 Gramatika v ELF

Soubor ve formátu ELF popisující síť obsahuje hlavičku následovanou posloupností uzlů a posloupností hran. Uzly jsou slova a hrany jsou přechody mezi slovy. Přechody mezi slovy mohou být ohodnoceny, ohodnocení musí být nezáporné. Používá se penalizačního ohodnocení (součet pravděpodobností přechodů může být roven jedné, ale nemusí). Případnou normalizaci půjde zvolit při kompilaci. Uzly a hrany jsou očíslovány a první definiční řádek musí obsahovat počet uzlů a hran. Obvykle uzlům přísluší pojmenování, a tedy reprezentují slova a hrany reprezentují přechody mezi slovy s danou pravděpodobností v *log10* formátu.

Každý blok může obsahovat řádek komentáře začínající //, zbytek řádky bude ignorován. Všechny řádky neobsahující komentáře obsahují položky oddělené mezerami. Položky obsahují jméno položky skládající se z

malých a velkých znaků abecedy a číslic, jsou následovány oddělovačem (znak '=' nebo ' ') a hodnotou položky. Všechna písmena ve jménu položky jsou významová. Na základě konvence se malých písmen ve jménech užívá pro nepovinné položky a velkých písmen pro povinné položky. Hlavička musí obsahovat verzi ELF. Hlavička je ukončena řádkou obsahující počet uzlů a hran v grafu. V souboru ELF se rozlišují malá a velká písmena.

ELF tedy vždy obsahuje:

Jako první se v souboru uvádí verze ELF

```
VERSION=1.0
```

Následuje počet uzlů a hran

```
N=10 L=13
```

Zde se pak umisťuje blok uzlů.

- Za *I* se zapisuje číslo uzlu,
- za *W* se píše jméno uzlu,
- za *t* se píše typ uzlu,
- za *s* se píše, zda bude uzel kompilován se vstupem *sil*. Možné zápisy jsou *s=Y* (ano) a *s=N* (ne), pokud se neuvede, tak se apriori předpokládá nastavení *s=Y*,
- za *p* se píše, zda jméno uzlu je již foneticky transkribováno. Možné zápisy jsou *p=Y* (ano) a *p=N* (ne), pokud se neuvede tak se apriori předpokládá nastavení *p=N*,
- za *c* se píše pravý kontext. Možné zápisy jsou *c=foném*. Například *c=A* znamená, že pravý kontext má být uvažován s á (dlouhým a) **(Pozn. 5)**
- za *a* se zapisují tagy (anotace). Jednomu uzlu může příslušet více anotací.

Možné typy uzlů jsou následující:

- *t=Begin* označuje uzel, který je vstupním uzlem gramatiky. Jméno uzlu je získáno ze speciálního tagu příslušejícího k pravidlu <*BEGIN_GRAMMAR*>. V případě jeho nepřítomnosti je tento uzel pojmenován *BeginX*, kde *X* vyjadřuje pořadové číslo uzlu tohoto typu v grafu.
- *t=End* označuje uzel, který je výstupním uzlem gramatiky. Jméno uzlu je získáno ze speciálního tagu příslušejícího k pravidlu <*END_GRAMMAR*>. V případě jeho nepřítomnosti je tento uzel pojmenován *EndX*, kde *X* vyjadřuje pořadové číslo uzlu tohoto typu v grafu.
- *t=ExternIn* označuje uzel, který je vstupním uzlem z jiné gramatiky. Jméno uzlu je získáno ze speciálního tagu příslušejícího k pravidlu <*EXTERN_IN_GRAMMAR*>. V případě jeho nepřítomnosti je tento uzel pojmenován *ExternInX*, kde *X* vyjadřuje pořadové číslo uzlu tohoto typu v grafu.
- *t=ExternOut* označuje uzel, který je výstupním uzlem do jiné gramatiky. Jméno uzlu je získáno ze speciálního tagu příslušejícího k pravidlu <*EXTERN_OUT_GRAMMAR*>. V případě jeho nepřítomnosti je tento uzel pojmenován *ExternOutX*, kde *X* vyjadřuje pořadové číslo uzlu tohoto typu v grafu.
- *t=Normal* označuje uzel, který vzniknul ze symbolu (např. *test*).
- *t=Null* označuje uzel, který je zaveden z pomocných důvodů při kompilaci.
- *t=Virtual* označuje uzel, který je podobný uzlu typu *t=Null*, ale je bezkontextový. Tento uzel vzniká při vytváření *mumble* modelu nebo je ho možno přímo vložit.
- *t=Mumble* označuje uzel, který vznikl při konstrukci *mumble* modelu.

Pokud se neuvede typ uzlu, použije se typ *t=Normal*.

Je nutno zadávat parametry v daném pořadí. Například:

```
I=0    W={Begin1}  t=Begin     s=Y p=N
I=1    W={symbol}  t=Normal    s=Y p=N a={anotace1} a={anotace2}
I=2    W={}        t=Null      s=Y p=N
```

Pravidlo <*SIL*> se zavádí jako normální uzel s tím, že se tento uzel jmenuje <*SIL*>. Jinak řečeno, platí *W={}*.

Následuje blok popisující hrany. Za *J* se zapisuje číslo hrany, za *S* se píše číslo uzlu, ze kterého hrana vychází, za *E* se píše číslo uzlu, ve kterém hrana končí, za *l* se zapisuje pravděpodobnost přechodu.

```
J=0    S=0    E=1    l=0.000
J=1    S=1    E=2    l=-0.301
```

3.1 Příklady

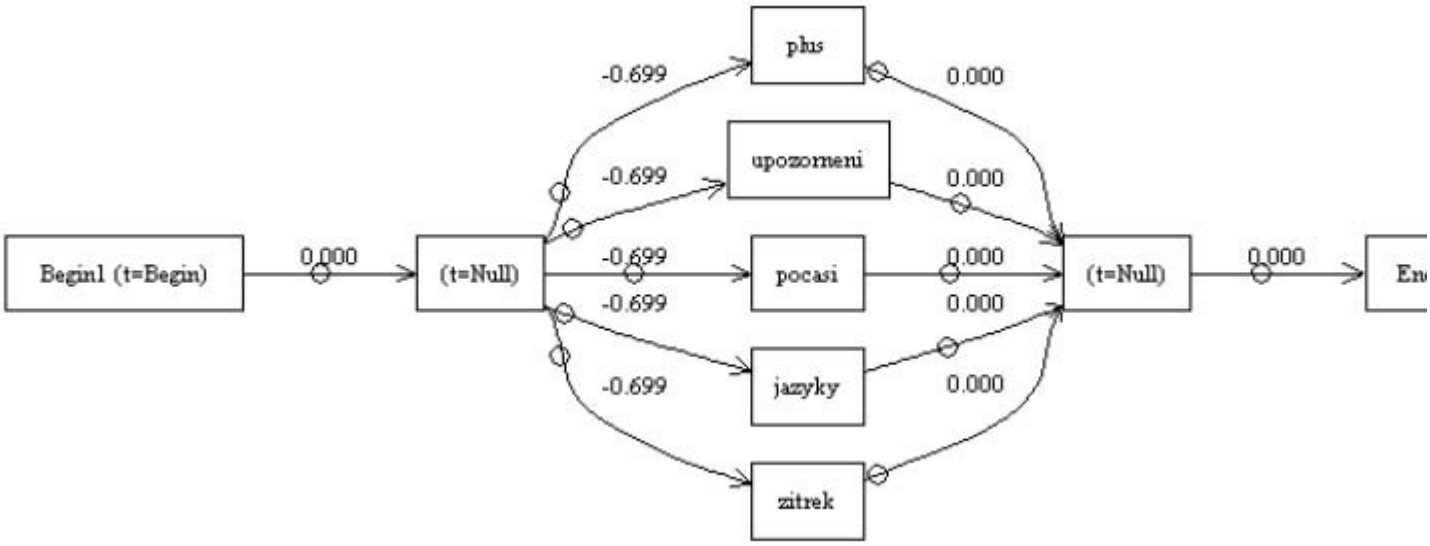
Následující příklady mají penalizační hodnoty normalizované.

3.1.1 Příklad 1

Nyní si uvedeme, jak bude vypadat zkompilovaný `Příklad_1.esgf`. Zde si musíme uvědomit, že obě verze grafu *v1* i *v2* budou mít stejný ELF soubor.

Příklad_1.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_1
// Number of attached lexical trees: 0

VERSION=1.0

// Size line

N=9 L=12

// Node definitions

I=0    W={Begin1}            t=Begin      s=Y p=N
I=1    W={}                  t=Null       s=Y p=N
I=2    W={}                  t=Null       s=Y p=N
I=3    W={plus}              t=Normal     s=Y p=N
I=4    W={upozornění}        t=Normal     s=Y p=N
I=5    W={počasí}            t=Normal     s=Y p=N
I=6    W={jazyky}            t=Normal     s=Y p=N
I=7    W={zítřek}            t=Normal     s=Y p=N
I=8    W={End1}              t=End        s=Y p=N

// Link definitions

J=0    S=1    E=3    l=-0.699
J=1    S=3    E=2    l=0.000
J=2    S=1    E=4    l=-0.699
J=3    S=4    E=2    l=0.000
J=4    S=1    E=5    l=-0.699
J=5    S=5    E=2    l=0.000
J=6    S=1    E=6    l=-0.699
J=7    S=6    E=2    l=0.000
J=8    S=1    E=7    l=-0.699
J=9    S=7    E=2    l=0.000
J=10   S=2    E=8    l=0.000
J=11   S=0    E=1    l=0.000
```



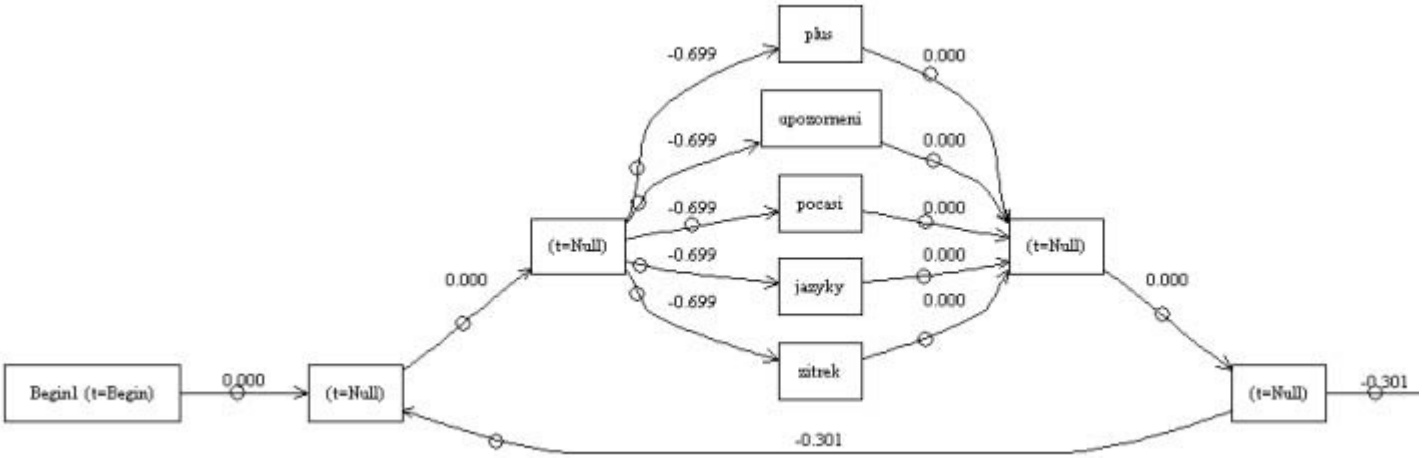
3.1.2 Příklad 2

Nyní si uvedeme jak bude vypadat zkompilovaný `Příklad_2.esgf`.

Příklad_2.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_2
// Number of attached lexical trees: 0

VERSION=1.0

// Size line

N=11 L=15

// Node definitions

I=0    W={Begin1}            t=Begin      s=Y p=N
I=1    W={}                  t=Null       s=Y p=N
I=2    W={}                  t=Null       s=Y p=N
I=3    W={plus}              t=Normal     s=Y p=N
I=4    W={upozornění}        t=Normal     s=Y p=N
I=5    W={počasí}            t=Normal     s=Y p=N
I=6    W={jazyky}            t=Normal     s=Y p=N
I=7    W={zítřek}            t=Normal     s=Y p=N
I=8    W={}                  t=Null       s=Y p=N
I=9    W={}                  t=Null       s=Y p=N
I=10   W={End1}              t=End        s=Y p=N

// Link definitions

J=0    S=1    E=3    l=-0.699
J=1    S=3    E=2    l=0.000
J=2    S=1    E=4    l=-0.699
J=3    S=4    E=2    l=0.000
J=4    S=1    E=5    l=-0.699
J=5    S=5    E=2    l=0.000
J=6    S=1    E=6    l=-0.699
J=7    S=6    E=2    l=0.000
J=8    S=1    E=7    l=-0.699
J=9    S=7    E=2    l=0.000
J=10   S=9    E=8    l=-0.301
J=11   S=8    E=1    l=0.000
J=12   S=2    E=9    l=0.000
J=13   S=9    E=10   l=-0.301
```

```
J=14    S=0     E=8     l=0.000
```



3.1.3 Příklad 3

Nyní si uvedeme jak bude vypadat zkompilovaný `Příklad_3.esgf`.

Příklad_3.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_3
// Number of attached lexical trees: 0

VERSION=1.0

// Size line

N=15 L=19

// Node definitions

I=0     W={Begin1}              t=Begin      s=Y p=N
I=1     W={}                    t=Null       s=Y p=N
I=2     W={<SIL>}              t=Normal     s=Y p=N
I=3     W={}                    t=Null       s=Y p=N
I=4     W={}                    t=Null       s=Y p=N
I=5     W={}                    t=Null       s=Y p=N
I=6     W={hned}                t=Normal     s=Y p=N
I=7     W={dnes}                t=Normal     s=Y p=N
I=8     W={zítra}               t=Normal     s=Y p=N
I=9     W={}                    t=Null       s=Y p=N
I=10    W={}                    t=Null       s=Y p=N
I=11    W={}                    t=Null       s=Y p=N
I=12    W={<SIL>}              t=Normal     s=Y p=N
I=13    W={}                    t=Null       s=Y p=N
I=14    W={End1}                t=End        s=Y p=N

// Link definitions

J=0     S=1     E=2     l=-0.301
J=1     S=1     E=3     l=-0.301
J=2     S=2     E=3     l=0.000
J=3     S=4     E=6     l=-0.477
J=4     S=6     E=5     l=0.000
J=5     S=4     E=7     l=-0.477
J=6     S=7     E=5     l=0.000
J=7     S=4     E=8     l=-0.477
J=8     S=8     E=5     l=0.000
J=9     S=10    E=9     l=-0.301
J=10    S=9     E=4     l=0.000
```

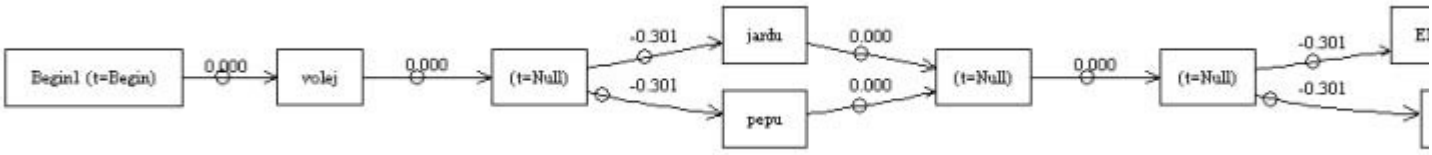
```
J=11    S=5     E=10    l=0.000
J=12    S=11    E=12    l=-0.301
J=13    S=11    E=13    l=-0.301
J=14    S=12    E=13    l=0.000
J=15    S=13    E=14    l=-0.301
J=16    S=10    E=11    l=-0.301
J=17    S=3     E=9     l=-0.301
J=18    S=0     E=1     l=0.000
```

3.1.5 Příklad 5 Vkládání gramatik

Nyní si uvedeme jak bude vypadat zkompilovaný `Příklad_5_Hlavní.esgf`.

Příklad_5_Hlavní.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_5_Hlavní
// Number of attached lexical trees: 0

VERSION=1.0

// Size line

N=9 L=9

// Node definitions

I=0     W={Begin1}              t=Begin     s=Y p=N
I=1     W={volej}               t=Normal    s=Y p=N
I=2     W={}                    t=Null      s=Y p=N
I=3     W={}                    t=Null      s=Y p=N
I=4     W={jardu}               t=Normal    s=Y p=N
I=5     W={pepu}                t=Normal    s=Y p=N
I=6     W={}                    t=Null      s=Y p=N
I=7     W={EL_1}                t=ExternOut s=Y p=N
I=8     W={End1}                t=End       s=Y p=N

// Link definitions

J=0     S=2     E=4     l=-0.301
J=1     S=4     E=3     l=0.000
J=2     S=2     E=5     l=-0.301
J=3     S=5     E=3     l=0.000
J=4     S=6     E=7     l=-0.301
J=5     S=6     E=8     l=-0.301
J=6     S=3     E=6     l=0.000
J=7     S=1     E=2     l=0.000
J=8     S=0     E=1     l=0.000
```

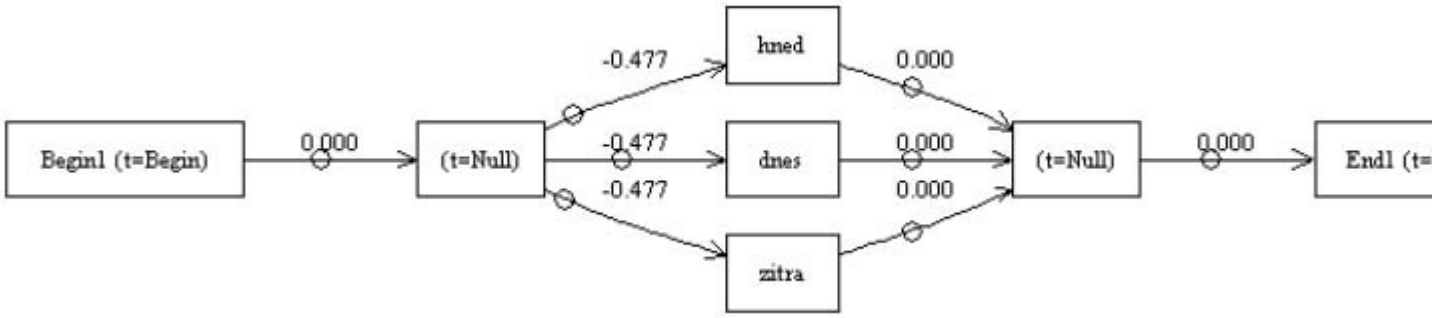


Nyní si uvedeme jak bude vypadat zkompilovaný `Příklad_5_Externí.esgf`.

Příklad_5_Externí.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_5_Externí
// Number of attached lexical trees: 0

VERSION=1.0
```

```
// Size line

N=7 L=8

// Node definitions

I=0    W={Begin1}            t=Begin     s=Y p=N
I=1    W={}                  t=Null      s=Y p=N
I=2    W={}                  t=Null      s=Y p=N
I=3    W={hned}              t=Normal    s=Y p=N
I=4    W={dnes}              t=Normal    s=Y p=N
I=5    W={zítra}             t=Normal    s=Y p=N
I=6    W={End1}              t=End       s=Y p=N

// Link definitions

J=0    S=1    E=3    l=-0.477
J=1    S=3    E=2    l=0.000
J=2    S=1    E=4    l=-0.477
J=3    S=4    E=2    l=0.000
J=4    S=1    E=5    l=-0.477
J=5    S=5    E=2    l=0.000
J=6    S=2    E=6    l=0.000
J=7    S=0    E=1    l=0.000
```



3.1.6 Příklad 6

Nyní si uvedeme jak bude vypadat zkompilovaný `Příklad_6.esgf`.

Příklad_6.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_6
// Number of attached lexical trees: 0

VERSION=1.0

// Size line

N=16 L=17

// Node definitions

I=0    W={Begin1}            t=Begin     s=Y p=N
I=1    W={}                  t=Virtual   s=Y p=N
I=2    W={}                  t=Virtual   s=Y p=N
I=3    W={}                  t=Mumble    s=Y p=N
I=4    W={Dnes}              t=Normal    s=Y p=N
I=5    W={je}                t=Normal    s=Y p=N
I=6    W={ale}               t=Normal    s=Y p=N
I=7    W={pěkně.}            t=Normal    s=Y p=N
I=8    W={<SIL>}            t=Normal    s=Y p=N
I=9    W={Že}                t=Normal    s=Y p=N
I=10   W={mám}               t=Normal    s=Y p=N
```

```
I=11   W={pravdu?}           t=Normal    s=Y p=N
I=12   W={}                  t=Virtual   s=Y p=N
I=13   W={}                  t=Virtual   s=Y p=N
I=14   W={}                  t=Mumble    s=Y p=N
I=15   W={End1}              t=End       s=Y p=N

// Link definitions

J=0    S=2    E=1    l=-0.301
J=1    S=1    E=3    l=-10.000
J=2    S=3    E=2    l=0.000
J=3    S=13   E=12   l=-0.301
J=4    S=12   E=14   l=-20.000
J=5    S=14   E=13   l=0.000
J=6    S=13   E=15   l=-0.301
J=7    S=11   E=12   l=0.000
J=8    S=10   E=11   l=0.000
J=9    S=9    E=10   l=0.000
J=10   S=8    E=9    l=0.000
J=11   S=7    E=8    l=0.000
J=12   S=6    E=7    l=0.000
J=13   S=5    E=6    l=0.000
J=14   S=4    E=5    l=0.000
J=15   S=2    E=4    l=-0.301
J=16   S=0    E=1    l=0.000
```

3.1.7 Příklad 7

Nyní si uvedeme jak bude vypadat zkompilovaný příklad na lexikální stromy. Ukážeme si užití parametrů *s,p* a *c*.

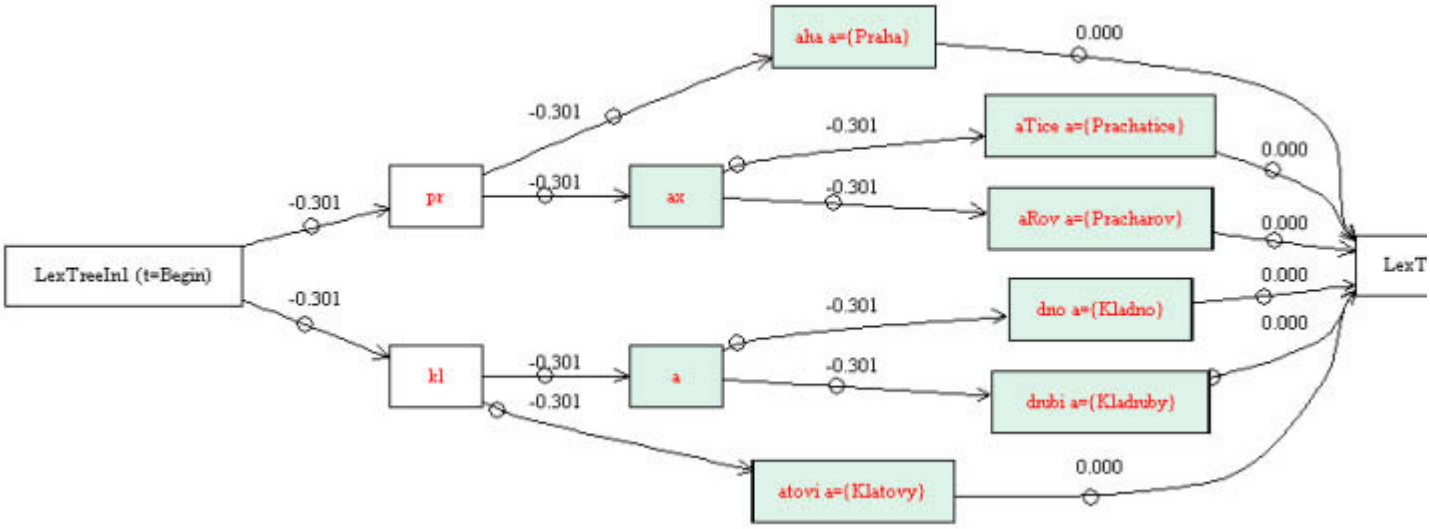
Příklad_7.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_7
// Number of attached lexical trees: 2

VERSION=1.0

// Size line

N=18 L=12

// Node definitions

I=0    W={Begin1}            t=Begin     s=Y p=N
I=1    W={LexTreeOut1_1}     t=ExternOut s=Y p=N
I=2    W={LexTreeIn1_1}      t=ExternIn  s=Y p=N
I=3    W={End1}              t=End       s=Y p=N
I=4    W={Begin2}            t=Begin     s=Y p=N
I=5    W={pan}               t=Normal    s=Y p=N
I=6    W={LexTreeOut2_1}     t=ExternOut s=Y p=N
I=7    W={LexTreeIn2_1}      t=ExternIn  s=Y p=N
I=8    W={cestuje}           t=Normal    s=Y p=N
I=9    W={z}                 t=Normal    s=Y p=N
I=10   W={města}             t=Normal    s=Y p=N
I=11   W={LexTreeOut1_2}     t=ExternOut s=Y p=N
I=12   W={LexTreeIn1_2}      t=ExternIn  s=Y p=N
I=13   W={do}                t=Normal    s=Y p=N
I=14   W={města}             t=Normal    s=Y p=N
I=15   W={LexTreeOut1_3}     t=ExternOut s=Y p=N
I=16   W={LexTreeIn1_3}      t=ExternIn  s=Y p=N
I=17   W={End2}              t=End       s=Y p=N

// Link definitions
```

```
J=0    S=2    E=3    l=0.000
J=1    S=0    E=1    l=0.000
J=2    S=14   E=15   l=0.000
J=3    S=13   E=14   l=0.000
J=4    S=12   E=13   l=0.000
J=5    S=10   E=11   l=0.000
J=6    S=9    E=10   l=0.000
J=7    S=8    E=9    l=0.000
J=8    S=7    E=8    l=0.000
J=9    S=5    E=6    l=0.000
J=10   S=16   E=17   l=0.000
J=11   S=4    E=5    l=0.000
```

Příklad_7_LexTree1.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_7
// Lexical tree formula name: města
// Lexical tree attachements number: 3

VERSION=1.0

// Size line

N=12 L=16

// Node definitions

I=0    W={LexTreeIn1}        t=Begin     s=Y p=N
I=1    W={pr}                t=Normal    s=Y p=Y c=a
I=2    W={aha}               t=Normal    s=N p=Y a={Praha}
I=3    W={ax}                t=Normal    s=N p=Y c=a
I=4    W={aTice}             t=Normal    s=N p=Y a={Prachatice}
I=5    W={aRov}              t=Normal    s=N p=Y a={Prachařov}
I=6    W={kl}                t=Normal    s=Y p=Y c=a
I=7    W={a}                 t=Normal    s=N p=Y c=d
I=8    W={dno}               t=Normal    s=N p=Y a={Kladno}
I=9    W={drubi}             t=Normal    s=N p=Y a={Kladruby}
I=10   W={atovi}             t=Normal    s=N p=Y a={Klatovy}
I=11   W={LexTreeOut1}       t=End       s=Y p=N

// Link definitions

J=0    S=1    E=2    l=-0.301
J=1    S=3    E=4    l=-0.301
J=2    S=3    E=5    l=-0.301
J=3    S=1    E=3    l=-0.301
J=4    S=0    E=1    l=-0.301
J=5    S=7    E=8    l=-0.301
J=6    S=7    E=9    l=-0.301
J=7    S=6    E=7    l=-0.301
J=8    S=6    E=10   l=-0.301
J=9    S=0    E=6    l=-0.301
J=10   S=2    E=11   l=0.000
J=11   S=4    E=11   l=0.000
J=12   S=5    E=11   l=0.000
J=13   S=8    E=11   l=0.000
J=14   S=9    E=11   l=0.000
J=15   S=10   E=11   l=0.000
```



Příklad_7_LexTree2.elf

```
// Eris Lattice Format generated by ESGF Compiler s038 from Feb 26 2002
//
// Source grammar: Příklad_7
// Lexical tree formula name: jména
// Lexical tree attachements number: 1

VERSION=1.0

// Size line

N=17 L=23

// Node definitions

I=0    W={LexTreeIn2}        t=Begin     s=Y p=N
I=1    W={aleS}              t=Normal    s=Y p=Y
I=2    W={praZAk}            t=Normal    s=Y p=Y a={Aleš Pražák}
I=3    W={novosad}           t=Normal    s=Y p=Y a={Aleš Novosad}
I=4    W={filip}             t=Normal    s=Y p=Y
I=5    W={jurCICek}          t=Normal    s=Y p=Y a={Filip Jurčíček}
I=6    W={l}                 t=Normal    s=Y p=Y c=u
I=7    W={uboS}              t=Normal    s=N p=Y
I=8    W={SmIdl}             t=Normal    s=Y p=Y a={Luboš Šmídl}
I=9    W={uDek}              t=Normal    s=N p=Y
I=10   W={mi}                t=Normal    s=Y p=Y c=l
I=11   W={ler}               t=Normal    s=N p=Y a={Luděk Müller}
I=12   W={ller}              t=Normal    s=N p=Y a={Luděk Müller}
I=13   W={jiRI}              t=Normal    s=Y p=Y
I=14   W={zahraDil}          t=Normal    s=Y p=Y a={Jiří Zahradil}
I=15   W={krUta}             t=Normal    s=Y p=Y a={Jiří Krůta}
I=16   W={LexTreeOut2}       t=End       s=Y p=N

// Link definitions

J=0    S=1    E=2    l=-0.301
J=1    S=1    E=3    l=-0.301
J=2    S=0    E=1    l=-0.602
J=3    S=4    E=5    l=0.000
J=4    S=0    E=4    l=-0.602
J=5    S=7    E=8    l=0.000
J=6    S=6    E=7    l=-0.301
J=7    S=10   E=11   l=-0.301
J=8    S=10   E=12   l=-0.301
J=9    S=9    E=10   l=0.000
J=10   S=6    E=9    l=-0.301
J=11   S=0    E=6    l=-0.602
```

```
J=12    S=13    E=14    l=-0.301
J=13    S=13    E=15    l=-0.301
J=14    S=0     E=13    l=-0.602
J=15    S=2     E=16    l=0.000
J=16    S=3     E=16    l=0.000
J=17    S=5     E=16    l=0.000
J=18    S=8     E=16    l=0.000
J=19    S=11    E=16    l=0.000
J=20    S=12    E=16    l=0.000
J=21    S=14    E=16    l=0.000
J=22    S=15    E=16    l=0.000
```

Pozn. 5
Zapisovat pravý kontext je zejména nutné v případě generování lexikálních stromů. Z důvodu implementace Elf compileru není možné zjistit u *A* strapů pravý kontext.

---

4 Literatura

[1] Java Speech Grammar Format Specification, Sun.
[2] IBM Speech Programmer's Guide : Chapter 6. Grammars, IBM.
[3] The HTK Book, Version 2.2, Entropic.
[4] Jak psát Eris.
[5] Síť pro dekodér, Eris.