Základy počítačové lingvistiky

Karel Pala
Klára Osolsobě

1. prosince 2001

Obsah

0.1 Úvod

Tento text je určen posluchačům filologických oborů na FF MU. Jeho cílem je uvést posluchače do základů počítačové lingvistiky a seznámit je s využíváním počítačů v jazykovědě, konkrétně v oblasti české morfologie, větné syntaxe i sémantiky.

V tomto smyslu lze předkládaný text pokládat rovněž za úvod do některých partií teoretické a počítačové lingvistiky.

Jde-li nám o využití počítačů v jazykovědě, musíme se také seznámit aspoň v základních obrysech s programovacím jazykem, který díky své orientaci na řešení problémů umožňuje pracovat na počítačích s formálními gramatikami a tak řešit experimentálně lingvistické problémy. Tímto programovacím jazykem je PROLOG, jehož standardní verze obvykle obsahují možnost přímo pracovat s gramatikami v notaci, která se velmi blíží aparátu užívanému běžně lingvisty při práci s (nekontextovými) gramatikami. Psát gramatiky v PROLOGU pak znamená de facto v něm programovat.

Náš výklad začínáme stručnou charakteristikou PROLOGU a jejím cílem je poskytnout čtenáři tu nejzákladnější představu o tom, co je PROLOG, a naznačit, jaký je jeho vztah k predikátové logice 1. řádu. V této souvislosti bychom rádi upozornili čtenáře i na to, že k dobrému porozumění těchto partií je nezbytné věnovat pozornost základům logiky (např. Berka, Mleziva, 1962).

Náš text neobsahuje úvod do práce s textovými editory a do zpracování textů vůbec. Jsme si velmi dobře vědomi, že v příručce pro posluchače filozofické fakulty by partie s tímto obsahem byla nepochybně potřebná, náš předmět zájmu je však natolik vyhraněný, že výklady o zpracování do našeho textu dost dobře nezapadají. Samostatná příručka věnovaná výhradně zpracování textů snad vznikne někdy později. Stejně tak by tento text mohl být rozšířen o popis práce s databázovými systémy (DBASE III, IV, FOXBASE, ORACLE apod.) a o příklady jejich aplikací v lingvistických oborech (tvoření bibliografií, souborů excerpt apod.), ale doufáme, že poučení v tomto směru si zvídavý čtenář najde sám v řadě k tomu určených příruček a manuálů.

Pokud jde o technické vybavení, s nímž v našem textu počítáme, předpokládáme, že čtenář bude mít k dispozici osobní počítače kompatibilní s IBM PC a pracující pod operačním systémem MS DOS. Měl by tedy být obeznámen s běžnými typy editorů, jako jsou T602, CSED, NORTON EDITOR, WORD 6.0 apod. U PROLOGU vycházíme z toho, že k dispozici jsou systémy jako SWI PROLOG, v. 2.9.0 a případně i některé lepší, např. SICSTUS PROLOG či snad dokonce ECLIPSE PROLOG. Rovněž předpokládáme základní znalost operačního systému MS DOS počínaje verzí 3. 2, k němuž běžně existují samostatné a snadno dostupné (i české) manuály.

0.2 Co je PROLOG?

PROLOG je programovací jazyk vysoké úrovně založený na predikátové logice 1. řádu a užívaný pro řešení problémů v různých vědních oblastech a mimo jiné v počítačovém zpracování přirozeného jazyka – pro výzkum v této oblasti byl také původně Colmerauerem navržen s původním názvem Q jazyk. Abychom porozuměli jeho podstatě, vysvětlíme nejprve, jak se

problémy, které nás mohou zajímat, v PROLOGU vyjadřují, tj. jak se v něm reprezentují.[1]

Problémy můžeme formulovat pomocí objektů a vztahů mezi nimi. Záleží však na tom, o jaké objekty půjde a jaké vztahy mezi objekty mohou existovat. Nejběžnějším a obecně dostupným prostředkem, v němž lze mluvit o objektech a vztazích mezi nimi, je pochopitelně přirozený jazyk – v našem případě čeština. Tak řekneme-li česky *Jan miluje Marii*, konstatujeme, že existují individuální objekty *Jan* a *Marie* a mezi nimi vztah *milování*, tj. v tomto případě, že Jan má rád Marii. Ve skutečnosti je celá situace poněkud složitější – přirozený jazyk umožňuje pomocí svých výrazů odkazovat k objektům v univerzu promluvy (tj. světě, o němž se v jazyce lze vyjadřovat) a tvrdit vztahy mezi objekty. Říkáme také, že výrazy přirozeného jazyka označují objekty a vztahy v univerzu. Je však potřeba vědět, jaké typy objektů a vztahů v našem univerzu promluvy existují a jak vlastně přiřazujeme výrazy jazyka objektům a vztahům v univerzu promluvy, tj. jak funguje přiřazení, které také nazýváme **označení** či **denotace**.

PROLOG v tomto směru nabízí standardní prostředky pro zápis významu výrazů přirozeného jazyka, takže umožňuje reprezentovat objekty a vztahy mezi nimi a ukládat je do paměti počítače a také se na ně dotazovat, tj. požadovat na PROLOGU, aby vztahy v jistém smyslu vyhodnocoval a také z jedněch vztahů odvozoval (logicky) jiné.

V PROLOGU tedy programujeme tak, že

- tvrdíme nějaké propozice o objektech a vztazích mezi nimi, přičemž propozicí zde rozumíme obecný význam oznamovací věty v přirozeném jazyce (zde v češtině),
- definujeme pravidla vyjadřující vztahy mezi objekty,
- ptáme se na objekty a vztahy mezi nimi.

Obrazně řečeno, PROLOG si lze představit jako svého druhu „sklad“ propozic a pravidel, který uložíme do počítače – systém PROLOG potom poskytuje způsob, jak na počítači z jedněch propozic vyvozovat propozice další. PROLOG je tedy inferenční (odvozovací) automat, který umožňuje z nějakých předpokladů vyvozovat platné závěry, a to pomocí počítače a ve formálním rámci predikátové logiky 1.řádu.

PROLOGU je přirozeně jedno, zda se dané propozice a pravidla týkají rozpisky automobilových součástek nebo objektů a vztahů existujících ve struktuře přirozeného jazyka, tj. např. vztahů mezi slovy, která tvoří třeba nějakou českou větu.

0.2.1 Reprezentace propozic v PROLOGU

Chceme-li v PROLOGU reprezentovat význam věty
(v-1) *Jan miluje Marii.*,
musíme si uvědomit, že se v ní mluví o dvou individuálních objektech – Janovi a Marii a o vztahu milování, který existuje mezi Janem a Marií, tj. o tom, že Jan má rád Marii. Věty jako (v-1) reprezentujeme v PROLOGU takto[2]:

[1] Následující výklad volně navazuje na text základní učebnice PROLOGU autorů Clocksina a Mellishe (1986).

[2] Pohled na vztahy mezi výrazy jazyka a objekty těmito výrazy označované, jak je uplatňován v PROLOGU, vychází z predikátové logiky 1. řádu a je tedy v podobě zde představené v jistém směru omezený. Skutečné

(c-1) `miluje(jan,marii).`
Platí přitom následující konvence:

- Jména všech objektů a vztahů musí začínat malým písmenem. Protože výrazy `jan` a `marie` jsou jména individuálních objektů, říkáme jim také individuální konstanty.
- Jméno vztahu se vždy píše jako *první* a za ním následují jména objektů uzavřená do jednoho páru kulatých závorek. Místo o jménech vztahů mluvíme častěji o predikátech a jména objektů nazýváme argumenty. Místo (c-1) můžeme také psát `m(a,b)` a pamatovat si, že `m` znamená `miluje`, `a` – `jan`, `b` – `marie`. Počet argumentů u jednoho predikátu není nijak zvlášť omezen – může jich být 15 – 30 v závislosti na typu použitého systému PROLOGU.
- Na konci výrazu, jako je (c-1), píšeme vždy tečku.
- Když definujeme vztahy mezi objekty, musíme dbát na pořadí argumentů. Rozdílu mezi českými větami
  (v-1) *Jan miluje Marii*
  a
  (v-2) *Marie miluje Jana*,
  odpovídá po řadě rozdíl mezi propozicemi zapsanými jako
  (c-1) `miluje(jan,marii).`
  a
  (c-2) `miluje(marie,jana).`

České věty
(v-3a) *Zlato je cenné.*
(v-3b) *Ideje jsou cenné.*
(v-4) *Anna je žena.*
(v-5a) *Anna pohrdá zlatem.*
(v-5b) *Anna pohrdá kariérou.*
(v-6) *Karel je bratr Anny.*
(v-7) *Anna nemiluje Karla.*
vypovídají o určitých objektech a jejich vlastnostech nebo o vztazích mezi objekty. U vlastních jmen jsme zvyklí, že označují určité konkrétní individuální objekty, nejasnosti však mohou vznikat u výrazů jako *zlato*. Můžeme se postavit na stanovisko, že výraz *zlato* označuje konkrétní individuální kus zlata, pak (v-3) má parafrázi
(v-3c) *Tento kus zlata je cenný.*
Je možná i obecnější interpretace, kterou bychom česky nejspíše formulovali takto:
(v-3d) *Minerál zvaný zlato je cenný.*

Z hlediska PROLOGU jsou přípustné obě tyto interpretace, musíme však být konzistentní a držet se té, pro kterou jsme se jednou rozhodli.

vztahy mezi výrazy jazyka a objekty jsou nepochybně složitější, a zejména je složitější univerzum promluvy. K zachycení této složitosti již predikátová logika 1. řádu nestačí, a proto je potřeba uchýlit se k aparátu logiky intenzionální (viz např. Materna, Pala, Zlatuška, 1989)

V PROLOGU je také jedno, jsou-li deklarované propozice pravdivé nebo ne. PROLOGU klidně můžeme sdělit propozici
(c-3) `král(michal,rusko).`,
jíž odpovídá nepravdivá česká věta
(v-7) *Michal je králem Ruska.*,
a PROLOG ji bude pokládat za pravdivou.

V tomto ohledu PROLOG předkládané propozice nijak neverifikuje (ani k tomu nemá vhodné prostředky), takže objekty a vztahy mohou sice být voleny libovolně, potom je však třeba dodržet potřebnou konzistenci.

Soubor propozic se v PROLOGU nazývá databáze. Databáze může ovšem vedle propozic obsahovat též pravidla, která potřebujeme pro řešení zadaného problému.

Naše propozice se mohou týkat nejen lidí a zlata, ale i prvků přirozeného jazyka, jako je čeština, a vztahů mezi nimi. V PROLOGU můžeme tedy bez obtíží vyjádřit českou větu
(v-8) *Slovo* **král** *je podstatné jméno.*
jako
(c-4) `n(král).`,
kde symbol `n`, jak již víme, označuje podstatné jméno.

Podobně věta
(v-9) *Česká věta se skládá z podmětu a přísudku.*
může mít v PROLOGU reprezentaci
(c-5a) `věta(po,přís).`
nebo v souladu s Chomským
(c-5b) `s(np1,vp).`
Povšimněme si, že (c-5a) a (c-5b) se nápadně podobají pravidlům (p-1) a (p-1a) uvedeným výše. Je tedy vidět, že tímto způsobem bychom mohli v PROLOGU reprezentovat řadu dalších propozic s lingvistickou interpretací. V dalším si ukážeme, že pro tento účel je v PROLOGU k dispozici speciální zápis, v němž můžeme zapisovat pravidla nekontextových gramatik (viz níže). Nekontextové gramatiky zapsané touto notací budeme v PROLOGU nazývat gramatikami vymezených klauzulí (DC gramatiky).

0.2.2 Otázky (zjišťovací)

Jakmile si v PROLOGU vytvoříme nějaký soubor propozic – tj. databázi, můžeme se na ně ptát. Otázka v PROLOGU se od propozice liší tak, že začíná speciální kombinací znaků tvořenou otazníkem a pomlčkou, tedy
(c-6) `?- miluje(marie,jana).`
Tato otázka odpovídá české tázací větě
(v-10a) *Je pravda, že Marie miluje Jana?*
resp. její jednodušší parafrázi
(v-10b) *Miluje Marie Jana?*
Otázka (c-6) je pro PROLOG povelem, aby začal prohledávat databázi, kterou jsme mu předtím zadali. Hledá propozice, které by se srovnaly s propozicí v otázce. Dvě propozice se srovnají, když se plně shodují jejich predikáty i jejich argumenty. Srovnáním se tedy rozumí plná shoda, proto např. platí `miluje = miluje`, nikoli však `miluje = miluji`. Podobně platí `m(a,b) =`

`m(a,b)`, nikoli však `m(a,b) = m(c,b) nebo m(a,b) = m(b,a)`.

Jakmile PROLOG najde v databázi propozici, která se srovná s otázkou, odpoví `yes (ano)`. Jestliže se v databázi taková propozice nenajde, odpověď zní `no (ne)`. Odpověď se vždy objevuje na obrazovce počítače na novém řádku pod zadanou otázkou, tedy:
(c-7) `?- miluje(marie,jana).`
`no`
`?-`
Vytvořme si nyní databázi obsahující propozice, které odpovídají českým větám (v-3) – (v-7).
(c-8) `cenné(ideje).`
(c-9) `cenné(zlato).`
(c-10) `žena(anna).`
(c-11a) `pohrdá(anna,zlato).`
(c-11b) `pohrdá(anna,kariéra).`
(c-12) `bratr(karel,anna).`
(c-13) `miluje(anna,karel).`

Nyní můžeme PROLOGU klást otázky a prolog na ně odpoví:

```
?- miluje(anna,zlato).
no
?- pohrdá(bratr, zlato).
no
?- žena(karel).
no
?- cenné(zlato).
yes
?- cenné(ideje).
yes
?-
```

Odpovědi na první tři otázky jsou jistě jasné. PROLOG nemůže odpovědět `yes` na otázku, u které nedošlo k plné shodě s příslušnou propozicí v databázi nebo pro kterou v databázi neexistují žádné propozice. Odpověď `no` má v PROLOGU obvykle význam *„pokud je mi známo, ne“*.

Zatím, jak patrno, dovedeme v PROLOGU kladením otázek dostat zpět informaci, kterou jsme zadali do databáze uložené v počítači. Jinak řečeno, PROLOG je s to odpovídat na zjišťovací otázky `no` nebo `yes`. To je pro nás ovšem dosti málo. Budeme chtít více, chceme dostávat odpovědi na doplňovací otázky jako
(v-11) *Kterými věcmi Anna pohrdá?*

0.2.3 Proměnné (otázky doplňovací)

Nyní nás zajímá, zda můžeme v PROLOGU reprezentovat doplňovací otázky jako
(v-12) *Čím pohrdá Anna?*
(v-13) *Co je cenné?*
nebo jinými slovy

(v-14) *Kterým objektem X pohrdá Anna?*
a dostávat na ně odpovědi.
Když klademe doplňovací otázky, nevíme, které objekty mohou stát za `X`, tj. místo tázacího zájmena, a chceme, aby PROLOG zjistil všechny možnosti.

Stejně jako v přirozeném jazyce ani v PROLOGU nemusíme vyjmenovávat všechny určité objekty, ale můžeme užít zástupných výrazů (tedy zájmen), které nahrazují určité objekty. Jména tohoto druhu nazýváme proměnné a rozlišujeme volné proměnné (u nich není známo, který objekt zastupují) a vázané proměnné (u nich je již znám objekt, který zastupují). Rozlišení jmen proměnných a jmen konkrétních objektů je v PROLOGU snadné, protože jména proměnných musí vždy začínat velkým písmenem, zatímco jména konkrétních objektů začínají malým písmenem.

Když PROLOG dostane otázku obsahující proměnnou, prochází propozice uložené v databázi a hledá objekt, který by proměnná mohla zastoupit.

Zeptáme-li se v PROLOGU podle (v-12)
(c-14a) `?- pohrdá(anna,X).`
nebo
(c-14b) `?- pohrdá(anna,Čím).`
nebo
(c-14c) `?- pohrdá(anna,Kterým_objektem).`
je to pro PROLOG vždy táž otázka, protože nezáleží na tom, jaké jméno proměnné zvolíme, podstatné je jen to, že jméno proměnné musí začínat velkým písmenem.

Na otázky (c-14a) – (c-14c) PROLOG po řadě odpoví
`X = zlato`
`X = kariéra`
`Čím = zlato`
`Čím = kariéra`
`Kterým_objektem = zlato`
`Kterým_objektem = kariéra`

Jak PROLOG k těmto odpovědím dospěl? Např. v (c-14a) je proměnná `X` volná. PROLOG prohledává databázi a hledá propozici, která by se srovnala s otázkou. Proměnná `X` stojí u predikátu `pohrdá` na místě 2. argumentu a v databázi existuje týž predikát s 2. argumentem `zlato`. PROLOG tedy srovnává `pohrdá(anna,X).` s `pohrdá(anna,zlato).` a proměnnou `X` váže na konkrétní objekt `zlato`, neboť jde vždy o 2. argument. Proto PROLOG odpovídá `X = zlato` a `X = kariéra`, neboť proměnná `X` může být vázána na cokoli, čím Anna pohrdá. PROLOG po vypsání odpovědi čeká na reakci uživatele: stiskne-li uživatel klávesu `ENTER`, znamená to, že je spokojen s touto jedinou odpovědí, a PROLOG přestane dále hledat. Stiskneme-li klávesu středník „;“, PROLOG pokračuje v prohledávání databáze a snaží se vázat proměnnou `X` na nějaký další objekt, jímž Anna pohrdá.

Zeptáme-li se PROLOGU
(c-15) `?- pohrdá(Kdo, zlato).`,
chceme vědět, zda existuje někdo, kdo pohrdá zlatem. Z naší databáze vyplývá, že je to Anna. PROLOG tedy odpoví
`Kdo = anna`
Napíšeme-li za `anna` středník (;), PROLOG se pokusí najít všechny osoby, které v naší databázi pohrdají zlatem. Žádná další osoba pohrdající zlatem se však už v databázi nevyskytuje, proto PROLOG odpoví `no`.

Položíme-li otázku
(c-16) `?- pohrdá(Kdo,Čím).`
vyhledá PROLOG všechny osoby, které něčím pohrdají, a všechny objekty, jimiž někdo pohrdá, a na obrazovce vypíše
```
Kdo = anna;
Čím = zlato;
Čím = kariéra;
no
?-
```

0.2.4 Konjunkce a disjunkce

V PROLOGU lze klást i otázky po složitějších vztazích, např. větu
(v-15) *Kdo pohrdá zlatem a koho miluje Anna?*,
reprezentujeme v PROLOGU jako konjunkci dvou propozic
(c-17) `?- pohrdá(Kdo,zlato),miluje(anna,Koho).`
Čárka „,“ mezi predikáty `pohrdá` a `miluje` se čte *a*, je symbolem pro konjunkci a slouží k od-

dělování libovolného počtu propozic.

Co odpoví PROLOG na otázku (c-17)? Postupuje tak, že nejprve se snaží najít odpověď na první predikát. V našem případě uspěje, protože zlatem pohrdá Anna. Pokračuje tedy dál a v databázi zjistí, že Anna miluje Karla. Tím je úspěšně splněna i druhá propozice (cíl), takže PROLOG odpoví

```
Kdo = anna
Koho = karel.
```

Pokud není některý z cílů v konjunkci splněn, výsledná odpověď je vždy `no`.

V otázce se může vyskytovat i spojka *nebo*, např. ve větě
(v-16) *Pohrdá Anna zlatem nebo miluje Karla?*,
pak mluvíme o disjunkci a symbolem pro ni je středník „;“.

Odpovídající reprezentace disjunkce v PROLOGU vypadá takto:
(c-18) `pohrdá(anna,zlato);miluje(anna,karel).`
Spojce *nebo* odpovídá tedy středník „;“ a stačí, aby byl splněn aspoň jeden z cílů vyskytujících se v disjunkci – pak PROLOG odpoví

```
yes
?- .
```

0.2.5 **Pravidla** (implikace)

Představme si, že chceme v PROLOGU reprezentovat větu
(v-17) *Anna pohrdá všemi lidmi.*
Jeden způsob, ovšem poněkud těžkopádný, by mohl spočívat ve vypsání všech jednotlivých propozic
(c-19) `pohrdá(anna,karel).`
(c-20) `pohrdá(anna,petr).` ...
pro každou osobu, která je v naší databázi zachycena prostřednictvím svého jména.

Inteligentnější bude říci
(v-18a) *Anna pohrdá každým, kdo je osoba.*
nebo použít parafráze
(v-18b) *Jestliže je někdo člověk, pak Anna jím pohrdá.*
nebo poněkud formálněji
(v-18c) *Jestliže X je člověk, pak Anna pohrdá X.*
Odtud je již krůček k reprezentaci v PROLOGU
(c-21) `člověk(X) :- pohrdá(anna,X).`,
což je pravidlo v PROLOGU, tedy všeobecné tvrzení o objektech a vztazích mezi nimi (v predikátové logice 1. řádu bychom mluvili o výrazech obsahujících tzv. obecný kvantifikátor).

První část pravidla se v PROLOGU nazývá hlava, pak následuje symbol „:-“, který odpovídá české spojce *jestliže..., pak...*, a za ním je tělo pravidla. Pozorný čtenář si již jistě uvědomil, že větám jako (v-18b) nebo (v-18c) říkáme v logice implikace.

Tímto způsobem lze reprezentovat i poměrně dosti složité vztahy mezi objekty, zejména použijeme-li kromě spojky pro implikaci `:-` také spojek *a* (konjunkce) a *nebo* (disjunkce). Jako cvičení

si čtenář může zkusit zapsat v PROLOGU následující české věty:

(v-19) *Karel má rád každou ženu, která pohrdá zlatem a nekouří.*

(v-20) *X je bratr Y, jestliže X je muž a X a Y mají stejné rodiče.*

(v-21) *A může ukrást nějakou věc, jestliže A je zloděj a jestliže ta věc je cenná a A tu věc miluje.*

0.2.6 Mechanismus navracení

Pro získání základní představy o tom, jak PROLOG funguje, naznačíme stručně, co je v PROLOGU mechanismus navracení.

Mějme otázku
(v-22) *Existuje něco, čím Karel i Anna pohrdají?*

Tato otázka se skládá ze dvou cílů:

- najít nějaké `X`, jímž pohrdá Anna.
- zjistit, zda i Karel pohrdá tímto `X`, ať je to cokoli.

V PROLOGU to vyjádříme pomocí konjunkce
(c-22) `?- pohrdá(anna,X),pohrdá(karel,X).`
PROLOG se nejprve pokouší splnit první cíl (první člen konjunkce). Jakmile se cíl porovná s propozicí v databázi, PROLOG poznamená nalezené místo, kde došlo k srovnání, a snaží se splnit druhý cíl. Je-li splněn i druhý cíl, PROLOG poznamená místo, na kterém došlo k srovnání, a je tedy nalezeno řešení, které splňuje oba cíle.

Zapamatujme si však, že každý cíl má svůj vlastní ukazatel místa srovnání. Stane-li se, že druhý cíl není splněn, pak se PROLOG snaží znovu splnit předcházející (zde první) cíl. PROLOG přitom prohledává databázi kompletně pro každý cíl. Srovná-li se cíl s propozicí v databázi (a tím je splněn), pak PROLOG si toto místo poznamená pro případ pozdějšího nového plnění tohoto cíle. Potom již PROLOG hledá od takto nastaveného ukazatele místa, a nikoli od počátku databáze.

Na výše uvedené otázce (v-22) si nyní můžeme ukázat fungování mechanismu navracení (*backtracking*):

1. databáze se prohledává pro první cíl. Protože druhý argument je volná proměnná `X`, může se srovnat s čímkoli. První takto srovnaná propozice v databázi uvedené výše je `pohrdá(anna,zlato)`. Proměnná `X` se váže na `zlato`, a to všude, kde se v otázce vyskytuje. PROLOG poznamená místo v databázi, kde našel tuto propozici, takže se na ně může vrátit, bude-li potřeba znovu splnit týž cíl. Ještě si musíme zapamatovat, že `X` bylo vázáno právě zde, takže PROLOG může `X` „zapomenout“, potřebuje-li znovu splnit tento cíl.

2. nyní se databáze prohledává pro cíl `pohrdá(karel,zlato)`. Je tomu tak proto, že druhý cíl je `pohrdá(karel,X)` a `X` je vázáno na `zlato`. Můžeme se přesvědčit, že žádná taková propozice v naší databázi neexistuje. A protože cíl neuspěl, PROLOG se bude snažit znovu splnit cíl `pohrdá(anna,X)`, ale tentokrát hledá od místa, které si v databázi poznamenal. Předtím si musí proměnnou `X` „uvolnit“, aby ji mohl znovu s čímkoli srovnat.

3. protože jsme dosud nedosáhli konce databáze, prohledávání pokračuje a tak se srovná další propozice `pohrdá(anna, kariéra)`. Proměnná `X` je nyní vázána na `kariéra` a PROLOG si poznamená toto místo v databázi, pro případ dalšího znovusplňování.

4. PROLOG se opět snaží splnit druhý cíl stejným způsobem jako předtím. Protože v databázi již není propozice týkající se Karla a pohrdání, nedojde už k žádnému srovnání a druhý cíl nebude splněn. Nesplnění jednoho cíle v konjunkci znamená ovšem, že nebude splněna ani celá konjunkce.

5. PROLOG tedy na naši otázku odpoví `no`, zastaví se a bude čekat na naše další příkazy.

0.3 Morfologie

Chceme-li úspěšně popsat českou gramatiku jako celek, musíme se zabývat nejen syntaktickou, ale i morfologickou rovinou jazyka. Uvedeme zde nejprve některé základní pojmy z oblasti formální morfologie podle (MČ 2, 1986) a (HaJ, 1966).

0.3.1 Některé základní morfologické pojmy

Slovo jakožto jazyková jednotka bývá definováno různým způsobem na jednotlivých jazykových rovinách. Z hlediska formálního popisu je lze definovat jako posloupnost písmen oddělenou po obou stranách mezerami. Dále nás bude zajímat slovo jako jednotka morfologické roviny jazyka, jako jednotka morfologická a morfématická.

Morfologie je jazykovědná disciplína, která se zabývá morfematickou rovinou jazyka. Je to nauka o podobách, vzájemných vztazích a funkcích nejmenších segmentů jazyka nesoucích význam.

Morf je nejmenší znaková jednotka izolovaná při segmentaci promluvy. Morf je jednotka, kterou už dále nelze dělit. Morfy jsou konkrétními realizacemi abstraktních jazykových jednotek – morfémů.

Morfém je základní jednotka morfologického plánu jazyka. „Je to elementární jazykový znak, bilaterární jednotka (jednota označujícího a označovaného)“. Dále též:

1. Relační jednotka, určená vztahy k ostatním jednotkám.

2. Třída alomorfů (realizací morfémů = morfů).” (Erhart, 1984).

Každé ohebné slovo můžeme rozčlenit na dvě části (segmenty): lexikální část (tvarotvorný základ) – během flexe se nemění (kromě alternací kmene) a gramatickou část (slovotvorný formant) – během flexe se mění. Podobně hovoří Erhart o lexikálních a gramatických morfémech. V rámci této dichotomie existuje ještě další subklasifikace morfů. Uveďme nejprve klasifikaci podle (MČ 2, 1986).

Tvarotvorný základ představuje lexikální složka slovního tvaru a je buď jednoduchá (jednomorfová) – např. *knih-*, nebo lineárně členitá (vícemorfová) např. *knih-ař-stv-*, složená z kořene a slovotvorných afixů.

Tvarotvorný formant je gramatická složka slovního tvaru. Je rovněž buď jednomorfový - např. *-a, -ách*, nebo vícemorfový *-i-l-a*.

Tvarotvorný kmen je ta část slovního tvaru (popř. slova), která zbývá po odpojení jeho koncovky (popř. koncovek), nikoli tedy celého tvarotvorného formantu. (MČ 2, 1986).

Kmenotvorná přípona je segment, který rozšiřuje tvarotvorný základ na kmen, je však vzhledem k dichotomii tvarotvorný základ – tvarotvorný formant součástí tvarotvorného formantu.

Koncovky jsou tvaroslovné přípony (pádové, osobní a infinitivní) stojící v absolutním konci slovního tvaru (MČ 2, 1986).

Kořen je ta část slovního tvaru, která zbude po odtržení všech tvarotvorných i slovotvorných afixů.

Erhart v Základech jazykovědy užívá poněkud odlišnou terminologii. Poukazuje na možnost segmentace lexikálních morfémů na menší významové jednotky. Rozlišuje:

a) Primární deriváty = lexikální morfy („kmeny“) složené z kořene a derivačního afixu: *zedn-ík, lov-ec* apod.

b) Sekundární deriváty = lexikální morfy („kmeny“) složené z primárního derivátu („kmene“) a derivačního afixu: *učitel-k-a, lov-ec-ký* apod.

c) Složeniny = lexikální morfy složené ze dvou (resp. několika) kořenů nebo derivátů (a,b): *přínos, velko-město, kolo-běžk(a), ú-lov-ek, zlato-nosn(ý)* apod. („kmene“) a derivačního afixu (Erhart, 1984).

Gramatické morfémy se dělí na gramatické afixy vyskytující se pouze v kombinaci s kořeny a na gramatické pomocné kořeny, které mohou tvořit fonetická slova.

Pro naše potřeby je důležité dělení afixů podle postavení vzhledem ke kořeni na prefixy (předpony) stojící před kořenem a na sufixy (přípony) stojící za kořenem slova.

Pro přehlednost jsme vytvořili vlastní klasifikaci jednotlivých segmentů. Pro členění jmenných tvarů budeme používat rozdělení ohebného slova na kmen `km` a pádovou koncovku `t`. Jednoduché slovesné tvary rozčleníme na kořen `ko`, kmenotvornou příponu `kp` a koncovku `tv`.

Koncovka je nesamostatná část ohebného slovního tvaru, která se při flexi mění. Koncovky jsou sufixy nesoucí gramatické významy pádu, čísla a rodu u jmen, osoby, čísla, času, popř. druhu participia, čísla a jmenného rodu nebo infinitivu u sloves.

Kmen je ta část slovního tvaru substantiva, která zbývá po odtržení pádové koncovky. U sloves pod pojem kmen zahrnujeme komplex kořen + kmenotvorná přípona.

Kořen je název jednoho druhu submorfů. V předloženém textu budeme jako kořeny označovat segmenty vzniklé segmentací jednoduchých slovesných tvarů po odtržení osobní, participiální, či infinitivní koncovky a kmenotvorné přípony.

Za kmenotvornou příponu budeme pokládat segment izolovaný při členění jednoduchých slovesných tvarů (přehled slovesných kmenotvorných přípon viz u Komárka, 1987).

Tvaroslovné paradigma je soubor tvarů ohebného slova vyjadřujících systém jeho mluvnických kategorií .

Vzor je tvaroslovné paradigma reprezentované paradigmatem určitého slova, sloužící jako vzor paradigmat slov jiných (MČ 2, 1986).

Gramatický význam určuje vztahy mezi slovními druhy na úrovni syntagmatu. Funguje jako abstrakce pro jeden nebo více slovních druhů.

Gramatická forma je tvořena prostředky, jejichž pomocí se vyjadřuje určitý gramatický význam.

O gramatické kategorii mluvíme tehdy, jestliže je nějaký gramatický význam vyjadřován ustáleně nějakou gramatickou formou. Zatímco klasické popisy flexe se skládají z tabulkových přehledů pravidelných vzorů a ze seznamů výjimek a popisu jejich výskytu, algoritmický (dynamický popis) umožňuje zachytit jednotně celý systém a vytvořit počítačový model schopný generovat a rozpoznávat ohebné tvary slov.

Klasický popis české morfologie najdeme v tradičních českých mluvnicích, např. v České mluvnici (HaJ, 1966) nebo v (MČ 2, 1986). Tyto popisy jsou z hlediska formalizace nedostačující, poslouží nám však jako východisko k ní. Ukážeme si nejprve jednoduchý formální zápis a nastíníme problémy spojené s algoritmizací popisu české morfologie.

0.3.2 Substantiva

Nejdříve se budeme zabývat formální morfologií substantiv. Můžeme formulovat jednoduché pravidlo, které říká, že každé české podstatné jméno se skládá ze dvou složek: tvarotvorného základu (kmen) a tvarotvorného formantu (pádová koncovka).

Tuto skutečnost lze formálně zapsat následujícím způsobem:

(p-1) `sb` → `km t`
(p-2) `km` → `škol`
(p-3) `t` → `a`

Graficky: (g-1)

```
      sb

km        t

škol      a
```

Vidíme, že takto formulované pravidlo je příliš jednoduché. Vychází totiž z předpokladu, že česká flexe je pouhým kombinováním segmentů „tvarotvorný základ“ se segmenty „tvarotvorný formant“. Tento předpoklad je sice správný, flexi v češtině můžeme takto chápat, ovšem s určitým omezením. Jakému omezení podléhají „kombinace“ tvarotvorných základů substantiv a tvarotvorných formantů substantivních koncovek? Tímto omezením je vzor.

Vzor chápejme v této souvislosti jako množinu koncovek (tedy vlastně tvaroslovnou charakteristiku), které se pojí se substantivy téhož typu a vyjadřují u substantiv téhož typu gramatické kategorie čísla a pádu. Stejně bychom mohli definovat i vzory u jiných slovních druhů. Každý typ má zvláštní inventář koncovek (množiny koncovek tvoří průniky, které jsou někdy dosti rozsáhlé).

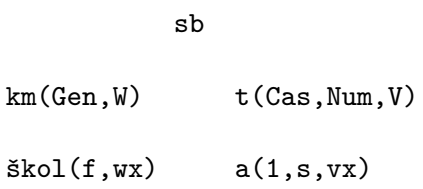
Jako příklad uvedeme koncovku ($-\emptyset$), která u substantiv rodu mužského životného tradičně zařazovaných ke vzorům *pán* nebo *muž* vyjadřuje nom. sg., u substantiv rodu mužského neživotného tradičně zařazovaných ke vzorům *hrad* nebo *stroj* – nom. a ak. sg., u substantiv rodu ženského tradičně zařazovaných ke vzoru *žena* – gen. pl., u substantiv rodu ženského tradičně zařazovaných ke vzorům *píseň* a *kost* – nom. a ak. sg. a u neuter tradičně zařazovaných ke vzoru *město* – gen. pl.

Z toho, co bylo řečeno, je patrné, že bychom měli do pravidel, jimiž popíšeme koncovky, zahrnout též gramatické významy, které vyjadřují. Postihneme tak homonymii českých substantivních koncovek.

Naznačené formulace zachytíme formálními pravidly. Každému substantivnímu tvarotvornému základu přiřadíme informaci o rodu a vzoru a každému substantivnímu tvarotvornému formantu informaci o odpovídajících gramatických významech kategorie pádu, čísla a vzoru. Docílíme tím jednak toho, že budeme moci vytvořit formální pravidla, dovolující kombinovat pouze tvarotvorné základy a tvarotvorné formanty, které k sobě patří, jednak bude výsledkem generování nejen substantivní tvar, ale i jeho gramatické významy.

(p-1a) `sb` → `km(Gen,W) t(Cas,Num,V)`
(p-2a) `km` → `škol(f,wx)`
(p-3a) `t` → `a(1,s,vx)`

Graficky: (g-1a)



Zkusme rozšířit slovník tvarotvorných základů substantiv rodu mužského životného (`mz`) skloňovaných podle vzoru *pán*. Uveďme např. tyto kmeny: *pán, pes, vlk, medvídek, občan.* Podívejme se nebo ještě lépe, zkusme odhadnout, jakých chyb bychom se dopustili, kdybychom tuto část české formální morfologie popsali algoritmicky následujícím způsobem:

```
(p-3b)    t →      0(1,s,v1)
(p-1b)    sb →     km(Gen,W) t(Pers,Num,V)
(p-2b)    km →     pán(mz,w1)
          km →     pán(mz,w1)
          km →     pes(mz,w1)
          km →     vlk(mz,w1)
          km →     občan(mz,w1)
          km →     medvídek(mz,w1)
(p-3b)    t →      0(1,s,v1)
          t →      0(1,s,v1)
          t →      a(2,s,v1)
          t →      u(3,s,v1)
          t →      ovi(3,s,v1)
          t →      a(4,s,v1)
          t →      u(6,s,v1)
          t →      ovi(6,s,v1)
          t →      em(7,s,v1)
          t →      i(1,p,v1)
          t →      ové(1,p,v1)
          t →      é(1,p,v1)
          t →      ů(2,p,v1)
          t →      ům(3,p,v1)
          t →      y(4,p,v1)
          t →      ech(6,p,v1)
          t →      ích(6,p,v1)
          t →      y(7,p,v1)
```

Podívejme se nyní na fragment DC gramatiky, kterou jsme napsali. Zjistíme, že kromě správných českých substantivních tvarů generuje také např. tvary jako:
vlk-i, vlk-ech
pes-a, pes-u, pes-ovi...
občan-i,..
medvídek-a,..medvídek-i,..medvídek-ích,...
Z toho plyne, že klasifikace vzorů, kterou uvádějí klasické gramatiky, je pro potřeby algoritmického popisu nedostatečná. Uvnitř vzorů, které známe ze školní praxe, musíme provést podstatně jemnější klasifikaci. Všimněme si typu chyb, které se vyskytly při generování substantivních tvarů.

Použité pravidlo nezachytilo skutečnost, že u některých slov dochází při skloňování ke změnám tvarotvorného základu, že slova skloňovaná klasicky podle jednoho vzoru mají pro některé pády alternativní koncovky, že se mění grafická podoba kmene atd.

Změny kmene a změny koncovky v české substantivní flexi

V české substantivní flexi dochází velmi často k některým změnám, na něž klasický popis formální morfologie sice upozorňuje, ke kterým ale neposkytuje řešení vhodné pro algoritmizaci. Alternace kmene, k nimž dochází při deklinaci i konjugaci, nebo alternativní koncovky, ať už jsou výslednicí historického vývoje flektivního typu nebo alternativní grafickou podobou koncovky, se v klasických mluvnicích popisují víceméně jako výjimky nebo jsou ponechány úplně stranou. Uveďme alespoň stručný přehled základních problémů způsobených zmíněnými jevy z hlediska algoritmického popisu.

Jsou to:

1) alternace –

a) samohláskové a tzv. vkladné *-e-*
(mráz – mrazu)(pes – psa)
b) souhláskové
(vlk – vlci)
c) Kombinace a) + b)
(dvůr – dvora – dvoře)
(medvídek – medvídka – medvídci)

2) alternativní koncovky –

a) vzniklé v důsledku rozpadu a míšení starých deklinačních typů *(páni – občané, hradu – lesa).*
b) plynoucí z ortografických zásad češtiny
(stroje – vězně, ženě – síle).

Pokusme se navrhnout vhodný algoritmus a odstranit chyby. Nejdříve musíme zjistit, kdy k alternacím dochází, přesně tyto případy definovat a na základě definic pak určit novou subklasifikaci vzorů.

ad 1a) Samohláskové alternace nelze popsat přesně formulovanými pravidly. Většinou jsou důsledkem historických hláskových změn. Můžeme ale říci asi tolik, že má-li slovo samohláskovou alternaci ve kmeni, pak platí, že jeden tvar je pro koncovku $-\emptyset$ a druhý pro ostatní koncovky.

Toto pravidlo bezesporu platí pro všechna maskulina a neutra a dále pro vkladné *„e“* u feminin. U feminin skloňovaných podle vzoru *žena* toto pravidlo platí s určitým omezením. K samohláskovým alternacím dochází totiž nejen v genitivu plurálu, ale i v lokálu a instrumentálu plurálu.

ad 1b) Souhláskové alternace jsou vázány na určitý typ koncovek. Jsou to koncovky *-i, -ích, -e.* Za souhláskové alternace pokládáme alternace velár a změnu *r-ř.* Nebudeme se zabývat alternacemi *d,t,n* u „tvrdých“ vzorů, který vzhledem k české ortografii nemají v písmu příslušnou realizaci (srov. *[hat-∅] – [haď-i]*). Naopak nás budou zajímat grafické alternace *d,t,n* srov. *[píseň] - [písně].*

ad 1c) K tomuto bodu nemusíme příliš mnoho dodávat. Je syntézou a) + b).

ad 2a) Tyto případy nelze přesně definovat pomocí formálních pravidel.

ad 2b) Jedná se především o výskyt koncovky *-ě, -ěm, -ěmi, -e, -em, -emi*. Zde již můžeme formulovat přesná pravidla. Podle zásad české ortografie se koncovky *-ě, -ěm, -ěmi* píší u měkkých vzorů po *-d,-t,-n,-b,-v,-f,-p,-m*, po ostatních grafémech píšeme *-e, -em, -emi*. Stejná je i distribuce koncovek *-ě, -e* u substantiv skloňovaných podle podvzoru *les* a u neuter vzoru *město*. Výjimku tvoří dvě skupiny slov, u kterých tato zásada neplatí. Jsou to jednak bývalé *n-kmeny*, jednak bývalé dlouhé *u-kmeny*.

Vzory

Vymezili jsme výše vzor z hlediska klasických popisů formální morfologie a objasnili jsme, jak chápeme tento pojem v rámci algoritmického popisu. Za vzor budeme pokládat přípustnou kombinaci tvarotvorných základů a tvarotvorných formantů.

Vzory a podvzory

Ukázali jsme si, že v rámci algoritmického popisu nevystačíme s klasickými čtrnácti českými vzory. Naznačili jsme obecně, na jaké problémy můžeme narazit. Podívejme se ještě jednou na příklady nesprávně generovaných substantivních tvarů. Zkusme se zamyslit nad analogickými případy u všech ostatních vzorů. Tak např. stejně jako je nesprávný tvar *občan-i*, je nesprávný i tvar *učitel-i*. Tak jako neexistuje tvar *vlk-i*, neexistuje ani tvar *matk-e* nebo *matk-ě*, atd. Vysvětlili jsme si gramatické pozadí jevů, které působí při algoritmickém popisu zmíněné nesnáze. Podívejme se na způsob, jak lze tyto nesnáze odstranit. Každý ze čtrnácti klasických českých vzorů, které známe ze školy, si můžeme představit jako tvaroslovnou charakteristiku (seznam koncovek slov skloňovaných podle tohoto vzoru). V rámci každé z těchto čtrnácti tvaroslovných charakteristik můžeme vyčlenit jádro, určitou podmnožinu koncovek, které nikdy nepůsobí ani změnu tvarotvorného základu, ani nealternují s jinou koncovkou. Zbytek koncovek lze pak podle druhu změn, které způsobují, rozdělit do jednotlivých podskupin – periferních vzorů (podvzory). Substantivní tvarotvorný základ nepatří potom k jedinému vzoru, ale vždy k jádru a několika periferním vzorům (podvzorům).

Ukažme si to na příkladech:
vzor : *-0,-a,-u,-ovi,-em,-i,-ové,-é,-ů,-ům,-ech,-ích,-y*

koncovky	jádro vzoru
–a	v1
–u	v1
–em	v1
–ů	v1
–ům	v1
–y	v1
koncovky	periferní vzory (podvzory)
–0	v1x
–i	v1y
–ové	v1a
–é	v1e
–ovi	v1f
–ech	v1z
–ích	v1q

Pak např. tvarotvorný základ *pes-* patří k podvzoru *v1x* a tvarotvorný základ *ps-* k jádru vzoru *v1* a k podvzorům *v1y, v1a, v1f, v1z.*

Tvarotvorný základ *vlk-* patří potom k jádru *v1* a dále k podvzoru *v1x, v1a, v1f* a tvarotvorný základ *vlc-* patří k podvzorům *v1y, v1q.* vzor žen-a: *-a,-y,-ě,-e,-u,-ou,-0,-ám,-ách,-ami*

koncovky	jádro vzoru žen-a
–a	v7
–y	v7
–u	v7
–ou	v7
koncovky	periferní vzory (podvzory)
–0	v7x
–e	v7a
–ě	v7b
–ám	v7e
–ách	v7f
–ami	v7g

Pak např. tvarotvorný základ *síl-* patří k jádru vzoru *v7* a k podvzoru *v7a* a tvarotvorný základ *sil-* k podvzorům *v7x, v7e, v7f, v7g.*

Uvedené podvzory jsme vyčlenili na základě alternací kmene a alternativních koncovek.

Tvarotvorné základy

Z toho, co bylo řečeno výše, vyplývá, že pro vybudování algoritmického popisu české substantivní deklinace vyjdeme z rozčlenění ohebného tvaru na dvě části: tvarotvorný základ a tvarotvorný formant. Tvarotvorný základ je nositelem gramatického významu jmenného rodu. Zároveň lze roztřídit tvarotvorné základy do tříd vzorů, které odpovídají novým vzorům vybudovaným pro potřeby algoritmického popisu.

Tvarotvorné formanty – koncovky

Součástí algoritmického popisu substantiv je úplný seznam pádových koncovek substantiv s uvedením gramatických významů kategorie pádu a čísla. Kromě toho je u každé koncovky třeba uvést gramatické významy pro vzor.

Podívejme se na souhrnné tabulky všech substantivních podvzorů, které jsme vyčlenili na základě alternací kmene a alternativních koncovek.

Substantiva tradičně řazená ke vzoru *pán*.

tab 1.

p.č.	v1	v1x	v1y	v1z	v1q	v1e	v1f	v1g	v1h
1.s.		-0							
2.s.	-a								
3.s.	-u,-ovi								
4.s.	-a								
5.s.								-e	-u
6.s.	-u,-ovi								
7.s.	-em								
1.p.			-i			-ové	-é		
2.p.	-ů								
3.p.	-ům								
4.p.	-y								
5.p.			-i			-ové	-é		
6.p.				-ech	-ích				
7.p.	-y								

Substantiva tradičně řazená ke vzoru *hrad*.

tab 2.

p.č.	v2	v2x	v2z	v2q	v2a	v2b	v2c	v2d	v2e
1.s.		-0							
2.s.							-u	-a	
3.s.	-u								
4.s.		-0							
5.s.								-e	-u
6.s.					-e	-ě			-u
7.s.	-em								
1.p.	-y								
2.p.	-ů								
3.p.	-ům								
4.p.	-y								
5.p.	-y								
6.p.			-ech	-ích					
7.p.	-y								

Substantiva tradičně řazená ke vzoru *muž*.

tab 3.

p.č.	v3	v3x	v3a	v3b	v3c	v3d	v3e	v3f	v3g
1.s.		-0							
2.s.			-e	-ě					
3.s.	-i				-ovi				
4.s.			-e	-ě					
5.s.						-i	-e		
6.s.	-i				-ovi				
7.s.			-em	-ěm					
1.p.								-ové	-é
2.p.					-ů				
3.p.					-ům				
4.p.			-e	-ě					
5.p.								-ové	-é
6.p.	-ích								
7.p.	-i								

Substantiva tradičně řazená ke vzoru *stroj*.

tab 4.

p.č.	v4	v4x	v4a	v4b	v4c
1.s.		-0			
2.s.			-e	-ě	
3.s.	-i				
4.s.			-e	-ě	
5.s.	-i				
6.s.	-i				
7.s.			-em	-ěm	
1.p.			-e	-ě	
2.p.					-ů
3.p.					-ům
4.p.			-e	-ě	
5.p.			-e	-ě	
6.p.	-ích				
7.p.	-i				

Substantiva tradičně řazená ke vzoru *předseda*.

tab. 5

p.č.	v5	v5y	v5z	v5q	v5g	v5h
1.s.	-a					
2.s.	-y					
3.s.	-ovi					
4.s.	-u					
5.s.	-o					
6.s.	-ovi					
7.s.	-ou					
1.p.		-i			-ové	-é
2.p.	-ů					
3.p.	-ům					
4.p.	-y					
5.p.		-i			-ové	-é
6.p.			-ech	-ích		
7.p.	-y					

Substantiva tradičně řazená ke vzoru *soudce*.

tab. 6

p.č.	v6	v6a	v6b	v6c	v6f
1.s.		-e	-ě		
2.s.		-e	-ě		
3.s.	-i			-ovi	
4.s.		-e	-ě		
5.s.		-e	-ě		
6.s.	-i			-ovi	
7.s.		-em	-ěm		
1.p.	-i				-ové
2.p.	-ů				
3.p.	-ům				
4.p.		-e	-ě		
5.p.	-i				-ové
6.p.	ích				
7.p.	-i				

Substantiva tradičně řazená ke vzoru *žena*.

tab. 7

p.č.	v7	v7x	v7a	v7b	v7c	v7d	v7e
1.s.	-a						
2.s.	-y						
3.s.			-e	-ě			
4.s.	-u						
5.s.	-o						
6.s.			-e	-ě			
7.s.	-ou						
1.p.	-y						
2.p.		-0					
3.p.					-ám		
4.p.	-y						
5.p.	-y						
6.p.						-ách	
7.p.							-ami

Substantiva tradičně řazená ke vzoru *růže*.

tab. 8

p.č.	v8	v8a	v8b	v8c	v8x
1.s.		-e	-ě		
2.s.		-e	-ě		
3.s.	-i				
4.s.	-i				
5.s.		-e	-ě		
6.s.	-i				
7.s.	-í				
1.p.		-e	-ě		
2.p.				-í	-0
3.p.	-ím				
4.p.		-e	-ě		
5.p.		-e	-ě		
6.p.	ích				
7.p.		-emi	-ěmi		

Substantiva tradičně řazená ke vzoru *píseň*.

tab. 9

p.č.	v9	v9x	v9a	v9b
1.s.		-0		
2.s.			-e	-ě
3.s.	-i			
4.s.		-0		
5.s.		-0		
6.s.	-i			
7.s.	-í			
1.p.			-e	-ě
2.p.	-í			
3.p.	-ím			
4.p.			-e	-ě
5.p.			-e	-ě
6.p.	ích			
7.p.			-emi	-ěmi

Substantiva tradičně řazená ke vzoru *kost*.

tab. 10

p.č.	v10	v10x
1.s.		-0
2.s.	-i	
3.s.	-i	
4.s.		-0
5.s.		-0
6.s.	-i	
7.s.	-í	
1.p.	-i	
2.p.	-í	
3.p.	-em	
4.p.	-i	
5.p.	-i	
6.p.	ech	
7.p.	mi	

Substantiva tradičně řazená ke vzoru *město*.

tab. 11

p.č.	v11	v11x	v11z	v11q	v11p	v11a	v11b	v11c
1.s.	-o							
2.s.	-a							
3.s.	-u							
4.s.	-o							
5.s.	-o							
6.s.						-e	-ě	-u
7.s.	-em							
1.p.	-a							
2.p.		-0						
3.p.	-ům							
4.p.	-a							
5.p.	-a							
6.p.			-ech	-ích	-ách			
7.p.	-y							

Substantiva tradičně řazená ke vzoru *moře*.

tab. 12

p.č.	v12	v12a	v12b	v12c	v12x
1.s.		-e	-ě		
2.s.		-e	-ě		
3.s.	-i				
4.s.	-i				
5.s.		-e	-ě		
6.s.	-i				
7.s.		-em	-ěm		
1.p.		-e	-ě		
2.p.				-í	-0
3.p.	-ím				
4.p.		-e	-ě		
5.p.		-e	-ě		
6.p.	ích				
7.p.	-i				

Substantiva tradičně řazená ke vzoru *stavení*.

tab. 13

p.č.	v13
1.s.	-í
2.s.	-í
3.s.	-í
4.s.	-í
5.s.	-í
6.s.	-í
7.s.	-ím
1.p.	-í
2.p.	-í
3.p.	-ím
4.p.	-í
5.p.	-í
6.p.	-ích
7.p.	-ími

Substantiva tradičně řazená ke vzoru *kuře*.

tab. 14

p.č.	v14	v14a	v14b
1.s.		-e	-ě
2.s.	-ete		
3.s.	-eti		
4.s.		-e	-ě
5.s.		-e	-ě
6.s.	-eti		
7.s.	-etem		
1.p.	-ata		
2.p.	-a2t		
3.p.	-atům		
4.p.	-ata		
5.p.	-ata		
6.p.	-atech		
7.p.	-aty		

0.3.3 Slovesa

Podobně jako substantiva lze analyzovat i slovesa. Na rozdíl od substantiv je při algoritmickém popisu formální morfologie českého slovesa výhodnější vycházet z některých lingvistických faktů, na něž jsme při popisu substantiv nebrali zřetel.

U podstatných jmen jsme vyšli z předpokladu, že každý substantivní tvar můžeme rozčlenit na dvě části, a to na kmen a koncovku. U sloves se nabízí potrojná segmentace na kořen, kmenotvornou příponu a koncovku. Podobně můžeme postupovat také v některých dalších případech. Tak např. při popisu slovotvorby nebo při popisu stupňování adjektiv.

Přehlédněme nyní formální zápis pravidel generování slovesných tvarů v češtině. Vycházíme přitom z analogie se substantivy.

```
(p-1c) v → ko kp t
(p-2c) ko→ uč
(p-3c) kp→ í
(p-4c) tv→ m
```

Graficky: (g-1c)

```
      v

ko    kp   tv

uč    í    m
```

Doplníme gramatické významy a typy segmentů

```
(p-1d) v   → ko(Wv)          kp(R)  t(Pers,Num,Mo,Vv)
(p-2d) ko  → uč(wv2aa)
(p-3d) kp  → í(r21)
(p-4d) tv  → m(1,s,in,vv2aa)
```

Graficky:(g-1d)

```
            v

ko(Wv)      kp(R)    tv(Pers,Num,Mo,Vv)

uč(wv2aa)   í(r21)   m(1,s,in,vv2aa)
```

Vzory

Vzor budeme chápat jako přípustnou kombinaci kořene, kmenotvorné přípony a koncovky.

Budeme postupovat následujícím způsobem: Vyjdeme z klasických slovesných tříd a vzorů. V češtině je klasifikace tříd a vzorů vybudována na:

a) rozdělení sloves podle kmenotvorné přípony kmene přítomného – třídy.

b) rozdělení sloves podle kmenotvorné přípony kmene minulého – vzory.

c) rozdělení podle zakončení kořene – některé vzory.

S každým slovesným kořenem se pojí určitý počet kmenotvorných přípon a určité podmnožiny koncovek. U každého slovesa potom rozeznáváme minimálně pět kmenů (kořen + kmenotvorná přípona) a pět subsystémů koncovek (koncovky indikativu prézenta, imperativu, participia l-ového, participia pasívního a infinitivu). Subklasifikaci slovesných vzorů jsme založili na podvzorech podle zmíněných subparadigmat. Tato subklasifikace není ovšem dostatečná.

1) Existují paralelní inventáře koncovek u jednoho a téhož „klasického vzoru“ (srov. *trp-∅, -me, -te bd-i ,-ěme, -ěte).*

2) I v rámci jednoho subparadigmatu se střídá u různých osob kmenotvorná přípona (srov.*nes-∅-u nes-e-š).*

Co tedy musíme vzít v úvahu, chceme-li vytvořit subklasifikaci slovesných kořenů s ohledem na vytváření vzorů sloves?

Zkoumejme nejprve jednotlivá dílčí slovesná paradigmata: řekli jsme si, že vyjdeme z pěti paradigmat.

1) Indikativu prézenta, které se dále rozpadne na další subparadigmata, a to na subparadigma 1.os. sg. a 3.os. pl. na jedné straně a ostatních osob na straně druhé. Názorněji si to můžeme ukázat formou tabulkového přehledu:

tab. 1

vv1a	*vv1b*	*vv1c*	*vv1d*	*vv1e*	*vv1f*
-e-	-ne-	-je-	-uje-	-í -	-á-
-š	-š	-š	-š	-š	-š
-0	-0	-0	-0	-0	-0
-me	-me	-me	-me	-me	-me
-te	-te	-te	-te	-te	-te

tab. 2

vv2a	*vv2b*	*vv2c*	*vv2d*	*vv2e*	*vv2aa*	*vv2ab*	*vv2ac*	*vv2ad*	*vv2ae*	*vv2af*
-0-	-n-	-j-	-uj-	-0-	-í -	-á-	-0-	-ej-	-ěj-	-aj-
-u	-u	-i	-i	-i	-m	-m				
-ou	-ou	-í	-í	-í			-í	-í	-í	-í

2) Imperativní paradigma se rozpadá na deset subparadigmat. Jejich distribuce závisí na fonetických a ortografických zákonitostech.

tab. 3

vv3a	*vv3b*	*vv3c*	*vv3d*	*vv3e*	*vv3f*	*vv3g*	*vv3h*	*vv3i*	*vv3j*
-0-	-ň-	-j-	-uj-	-ej-	-ěj-	-aj-	-0-	-0-	-n-
-0	-0	-0	-0	-0	-0	-0	-i	-i	-i
-me	-me	-me	-me	-me	-me	-me	-eme	-ěme	-ěme
-te	-te	-te	-te	-te	-te	-te	-ete	-ěte	-ěte

3) Paradigma l-ového participia, pro něž existuje jediný soubor koncovek, rozčleníme na sedm subparadigmat.

tab. 4

vv4a	*vv4b*	*vv4c*	*vv4d*	*vv4e*	*vv4f*	*vv4g*
-0-	-a-	-e-	-ě-	-nu-	-ova-	-i-
-l	-l	-l	-l	-l	-l	-l
-l	-l	-l	-l	-l	-l	-l
-la	-la	-la	-la	-la	-la	-la
-lo	-lo	-lo	-lo	-lo	-lo	-lo
-li	-li	-li	-li	-li	-li	-li
-ly	-ly	-ly	-ly	-ly	-ly	-ly
-ly	-ly	-ly	-ly	-ly	-ly	-ly
-la	-la	-la	-la	-la	-la	-la

4) Pasívní paradigma se rozpadne do sedmi podvzorů.

tab. 5

vv5a	*vv5b*	*vv5c*	*vv5d*	*vv5e*	*vv5f*	*vv5g*
-0-	-0-	-á-	-nu-	-a-	-0-	-ová-
-ěn	-en	-n	-t	-t	-t	-n
-ěn	-en	-n	-t	-t	-t	-n
-ěna	-ena	-na	-ta	-ta	-ta	-na
-ěno	-eno	-no	-to	-to	-to	-no
-ěni	-eni	-ni	-ti	-ti	-ti	-ni
-ěny	-eny	-ny	-ty	-ty	-ty	-ny
-ěny	-eny	-ny	-ty	-ty	-ty	-ny
-ěna	-ena	-na	-ta	-ta	-ta	-na

5) U infinitivního paradigmatu budeme rozlišovat deset podvzorů.

tab. 6

vv6a	*vv6b*	*vv6c*	*vv6d*	*vv6e*	*vv6f*	*vv6g*	*vv6h*	*vv6i*	*vv6j*
-0-	-á-	-a-	-í-	-nou-	-ova-	-i-	-ě-	-e-	-0-
-t	-t	-t	-t	-t	-t	-t	-t	-t	-i

Kořeny

Podle výše uvedené klasifikace vzorů bude každému českému slovesnému kořeni přiřazena tvaroslovná charakteristika (podvzor) pro generování tvarů určitých - prézenta (*wv1a* až *wv1f* a *wv2a* až *wv2e* nebo *wv2aa* až *wv2af*), tvarů imperativu (*wv3a* až *vw3j*) a pro tvary neurčité - participia l-ového (*wv4a* až *wv4g*), participia pasívního *wv5a* až *wv5g*) a infinitivu (*wv6a* až *wv6j*).

Koncovky

Sestavíme úplný seznam slovesných koncovek a provedeme jejich klasifikaci. Vzory koncovek a kořenů si navzájem odpovídají (viz. tab.).

Kmenotvorné přípony

Kmenotvorné přípony budeme číslovat a označíme je např. písmenem `r`, tedy `r1....rn`.

0.3.4 Česká morfologie v PROLOGU

PROLOG

PROLOG je, jak už bylo řečeno výše, programovací jazyk, kterého můžeme použít k řešení problémů, jež lze vyjádřit pomocí objektů a jejich vzájemných vztahů. Složitější vztahy mezi objekty můžeme popsat pomocí pravidel. PROLOG funguje interaktivně, to znamená, že uživatel může požádat určitým příkazem operační systém daného počítače (MS DOS o použití PROLOGU (viz. výše) a prostřednictvím obrazovky a klávesnice dále s PROLOGEM komunikovat. Na základě propozic, které má PROLOG k dispozici, odpoví na otázky, jež mu uživatel v pevně zadaném tvaru může klást.

Využití PROLOGU při popisu morfologie

Koncovky českých ohebných slov můžeme chápat jako vztahy objektů (morfů, písmen a gramatických významů, které za určitých okolností vyjadřují).

Propozice, že koncovka *-u* vyjadřuje u substantiv rodu mužského životného skloňovaných podle vzoru *pán* 3. nebo 6. pád jednotného čísla, můžeme v PROLOGU zapsat takto:
`/*c-1*/ t(u,3,s,v1)` → `[u].`
`/*c-2*/ t(u,6,s,v1)` → `[u].`
Jména objektů uzavřená do kulatých závorek se nazývají argumenty. Jméno vztahu nazýváme predikát. Soubor propozic v PROLOGU se nazývá databáze.

Jakmile máme nějaké propozice, můžeme se na ně v PROLOGU ptát. PROLOGOVSKÁ otázka je formálně totožná s propozicí s tím rozdílem, že před otázkou stojí prompt `?-`. Zadáme-li PROLOGU otázku, prohledává databázi, dokud nenajde propozici, jejíž predikát a argumenty jsou shodné s otázkou. Nenajde-li ji, odpoví `"no"`, což lze interpretovat jako `"ne, není mi o tom nic známo"`.

Tak např. položíme-li PROLOGU otázku nad databází obsahující výše uvedené propozice o koncovce *-u* a budeme-li chtít vědět, jaký pád může tato koncovka vyjadřovat, zeptáme se takto:
`/*c-3*/ ?- t(u,Cas,Num,v1).`
Slovně: Jaké jsou gramatické významy pádu a čísla vyjádřené koncovkou *-u* u substantiv skloňovaných podle vzoru *v1*? PROLOG odpoví následujícím způsobem:
`Cas=3`
`Num=s;`
`Cas=6`

```
Num=s;
no
```
Chceme-li se zeptat, zda neexistuje ještě nějaká další odpověď na naši otázku, napíšeme středník „;“.

PROLOGU můžeme ovšem klást i složitější otázky, např., zda platí jedna propozice a zároveň i druhá. Zeptáme-li se třeba, zda koncovka *-u*, může vyjadřovat jak dativ singuláru, tak lokál singuláru substantiv skloňovaných podle vzoru *v1*, můžeme otázku zapsat takto:

```
/*c-4/ ?- t(u,3,s,v1),t(u,6,s,v1).
yes
?-
```
Tímto způsobem lze v PROLOGU vyjádřit konjunkci dvou cílů. Mezi cíli píšeme *čárku „,“* a čteme ji *„a“*.

Je patrné, že bychom rádi uměli vyjadřovat i složitější vztahy, než jsme si dosud ukázali. PROLOG nám to samozřejmě umožňuje. V PROLOGU se k vyjádření složitějších vztahů, např.k vyjádření závislosti jedné propozice na propozicích ostatních, používá pravidel. Pravidlo je všeobecné tvrzení o objektech a jejich vztazích.

Doplňme si naši databázi o další propozice týkající se kmenů a vzorů substantiv.

```
/*c-5*/ km('pán',mz,w1).
/*c-6*/ vz(w1,v1).
```
Chceme nyní definovat vztah mezi propozicemi „koncovkami“ a „kmeny“. Můžeme, jak jsme si ostatně ukázali výše, říci, že substantivní tvar vzniká v češtině kombinací substantivního kmene a substantivní koncovky, jestliže jak kmen, tak i koncovka patří ke stejnému vzoru.

Pravidlo, jímž tuto skutečnost v PROLOGU reprezentujeme, se skládá z hlavy a těla. Hlava je spojena s tělem pomocí symbolu :-. Symbol :- se čte *jestliže, pak*, tj. jde o – implikaci. Použijeme-li gramatického preprocesoru, vestavěného v našem PROLOGU, dojde k nahrazení symbolu :- symbolem →. Uvedený příklad můžeme v PROLOGU zapsat takto:

```
/*c-7*/        sb(Km,T)       :-            vz(W,V),
                                            km(Km,Gen,W)
                                            t(T,Cas,Num,V).
```

Hlava pravidla popisuje vztah, který má být pravidlem definován. Tělo pravidla popisuje konjunkci cílů, které musí být postupně splněny, aby hlava byla pravdivá.

0.3.5 Přepis algoritmického popisu formální morfologie do jazyka PROLOG

Substantiva

Mějme následující databázi, která bude obsahovat:
1) Propozice
a) o koncovkách substantiv
b) o kmenech substantiv
c) o vzorech substantiv

2) Pravidla
Pravidlo tvoření ohebných substantivních tvarů, tj. kombinací konjunkcí) složek tvarotvorný základ (kmen), tvarotvorný formant (koncovka), pro které existuje společný vzor *vz*.

```
/*Pravidlo*/

/*c-8*/        sb(sb(Km,Ts))  →      vz(V,W),
                                     km(Km,Gen,W),
                                     t(T,Cas,Num,V).

/*Kmeny*/

/*c-9*/        km(km('pán'),mz,w1)      →      ['pán'].

/*Koncovky*/

/*c-10*/       t(t(0),1,s,v1)           →      [0].
               t(t(a),2,s,v1)           →      [a].
               t(t(a),4,s,v1)           →      [a].
               t(t(u),3,s,v1)           →      [u].
               t(t(u),6,s,v1)           →      [u].

/*Vzory*/

/*c-11*/       vz(w1,v1)                →      [w1,v1].
```

Využití predikátu `member`

Ukažme si, jak by vypadal kousek databáze v Prologu, tedy vlastně program generující ohebné tvary substantiv skloňovaných podle vzoru *pán*.

```
/*Pravidlo*/

/*c-12*/       sb(sb(Km,Ts))  →      vz(W,V),
                                     km(Km,W),
                                     t(Ts,Cas,Num,V).

/*Vzory*/

/*c-13*/       vz(w1,v1)                →      [w1,v1].
               vz(w1x,v1x)              →      [w1x,v1x].
               vz(w1y,v1y)              →      [w1y,v1y].
               vz(w1z,v1z)              →      [w1z,v1z].
               vz(w1q,v1q)              →      [w1q,v1q].
```

```
/*Kmeny*/

/*c-14*/      km(km('pán'),mz,w1)          →     ['pán'].
              km(km('pán'),mz,w1x)         →     ['pán'].
              km(km('pán'),mz,w1y)         →     ['pán'].
              km(km('pán'),mz,w1z)         →     ['pán'].
              km(km(vlk),mz,w1)            →     [vlk].
              km(km(vlk),mz,w1x)           →     [vlk].
              km(km(vlc),mz,w1y)           →     [vlc].
              km(km(vlc),mz,w1q)           →     [vlc].
              km(km(ps),mz,w1)             →     [ps].
              km(km(pes),mz,w1x)           →     [pes].
              km(km(ps),mz,w1y)            →     [ps].
              km(km(ps),mz,w1z)            →     [ps].
              km(km('občan'),mz,w1)        →     ['občan'].
              km(km('občan'),mz,w1x)       →     ['občan'].
              km(km('občan'),mz,w1y)       →     ['občan'].
              km(km('občan'),mz,w1z)       →     ['občan'].
              km(km('medvídk'),mz,w1)      →     ['medvídk'].
              km(km('medvídek'),mz,w1x)    →     ['medvídek'].
              km(km('medvídc'),mz,w1y)     →     ['medvídc'].
              km(km('medvídc'),mz,w1q)     →     ['medvídc'].
```

```
/*Koncovky*/

/*c-15*/      t(t(0),1,s,v1x)                 →          [0].
              t(t(a),2,s,v1)                  →          [a].
              t(t(a),4,s,v1)                  →          [a].
              t(t(u),3,s,v1)                  →          [u].
              t(t(u),6,s,v1)                  →          [u].
              t(t(ovi),3,s,v1)                →          [ovi].
              t(t(ovi),6,s,v1)                →          [ovi].
              t(t(em),7,s,v1)                 →          [em].
              t(t(i),1,p,v1y)                 →          [i].
              t(t('ů'),2,p,v1)                →          ['ů'].
              t(t('ům'),3,p,v1)               →          ['ům'].
              t(t(y),4,p,v1)                  →          [y].
              t(t(y),7,p,v1)                  →          [y].
              t(t(ech),6,p,v1z)               →          [ech].
              t(t('ích'),6,p,v1q)             →          ['ích'].
              t(t('ové'),1,p,v1f)             →          ['ové'].
              t(t('é'),1,p,v1e)               →          ['é'].
```

Je na první pohled patrné, že způsob ukládání kmenů do databáze je velice neefektivní. Každý kmen je v databázi uložen tolikrát, podle kolika podvzorů se skloňuje.

Podívejme se, jak funguje pravidlo, které popisuje generování substantivních tvarů. Nejprve PROLOG prohledává databázi a snaží se splnit první cíl, tj. dosadit za `vz(W,V)` nějaké konstanty `vz(w1,v1)`. Splnění dalšího cíle pak závisí na splnění cíle předchozího. Hledá se nějaké `km(Km,W)`, tedy kmen, kde se proměnná `Km` nahradí kmenem se vzorem označeným proměnnou `W`. `W` musí již být nahrazeno konkrétním podvzorem, tj. konstantou `w1` v souladu se splněním prvního cíle. Stejným způsobem se vyhledá i koncovka `t`. Proměnné `T, Cas, Num, V` se nahradí po řadě např. `u, 3, s, v1` nebo `em, 7, s, v1` atd., přičemž splnění tohoto cíle opět závisí na nahrazení proměnné `V` konstantou `v1` v souladu s prvním cílem, který byl splněn tak, že proměnná `V` byla nahrazena konstantou `v1`.

PROLOG nám nabízí možnost značně jednoduššího zápisu, s použitím predikátu member, jehož definici najdeme ve všech učebnicích PROLOGU.

```
/*Vzory*/


/*c-16*/  vz(w1,v1)                →      [w1,v1].
          vz(w1x,v1x)              →      [w1x,v1x].
          vz(w1y,v1y)              →      [w1y,v1y].
          vz(w1z,v1z)              →      [w1z,v1z].
          vz(w1q,v1q)              →      [w1q,v1q].
          vz(w1e,v1e)              →      [w1e,v1e].
          vz(w1f,v1f)              →      [w1f,v1f].

/*Kmeny*/


/*c-17*/  km(km('pán'),mz,[w1,w1x,w1y,w1z,w1e])   →      ['pán'].
          km(km(vlk),mz,[w1,w1x,w1e])             →      [vlk].
          km(km(vlc),mz,[w1y,w1q])                →      [vlc].
          km(km(pes),mz,[w1x])                    →      [pes].
          km(km(ps),mz,[w1,w1y,w1z])              →      [ps].
          km(km('občan'),mz,[w1,w1x,w1z,w1f])     →      ['občan'].
          km(km('medvídek'),mz,[w1x])             →      ['medvídek'].
          km(km('medvídk'),mz,[w1])               →      ['medvídk'].
          km(km('medvídc'),mz,[w1y,w1q])          →      ['medvídc'].


/*Koncovky*/


/*c-18*/  t(t(0),1,s,[v1x,v2x,v3x,v4x,v8x,v9x,v10x])                →   [0].
          t(t(a),2,s,[v1,v2d,v11])                                  →   [a].
          t(t(a),4,s,[v1])                                          →   [a].
          t(t(u),3,s,[v1,v2,v11])                                   →   [u].
          t(t(u),6,s,[v1,v2e,v11e])                                 →   [u].
          t(t(ovi),3,s,[v1,v3,v5,v6])                               →   [ovi].
          t(t(ovi),6,s,[v1,v3,v5,v6])                               →   [ovi].
          t(t(em),7,s,[v1,v2,v3a,v4a,v6a,v11,v12a])                 →   [em].
          t(t(i),1,p,[v1y,v3,v6,v10])                               →   [i].
          t(t('ů'),2,p,[v1,v2,v3,v4,v5,v6])                         →   ['ů'].
          t(t('ům'),3,p,[v1,v2,v3,v4,v5,v6,v11])                    →   ['ům'].
          t(t(y),4,p,[v1,v2,v5,v7])                                 →   [y].
          t(t(y),7,p,[v1,v2,v5])                                    →   [y].
          t(t(ech),6,p,[v1z,v2z,v5z,v11z])                          →   [ech].
          t(t('ích'),6,p,[v1q,v2q,v3,v4,v5q,v6,v8,v9,v11q,v12])     →   ['ích'].
          t(t('é'),1,p,[v1f,v3f])                                   →   ['é'].
          t(t('ové'),1,p,[v1e,v3e,v5e,v6e])                         →   ['ové'].
```

Jednou z podstatných vlastností PROLOGU je, že umožňuje pracovat se seznamy. Seznam je uspořádaná posloupnost prvků a může mít libovolnou délku. Seznam může být buď prázdný, nebo je to struktura, která má dvě komponenty: hlavu a tělo.

Máme např. nějaký seznam, kde X zastupuje hlavu a Y tělo. Můžeme jej zapsat jako `[X|Y]`. Tento seznam může obsahovat třeba podvzory klasického vzoru *pán* nebo s jeho pomocí můžeme zjednodušit popis koncovek, vyjadřuje-li např. jedna koncovka stejné gramatické významy u více vzorů:

`[w1,w1x,w1y,w1z,w1f]`
`[v1x,v2x,v3x,v4x,v8x,v9x,v10x]`
Seznamu `[v1x,v2x,v3x,v4x,v8x,v9x,v10x]` použijeme při definování koncovky ∅ českých substantiv vyjadřující gramatické významy *1. osoby singuláru* u klasických vzorů *pán – v1x, hrad – v2x, muž – v3x, stroj – v4x, růže – v8x, píseň – v9x, kost – v10x.*

PROLOG nám umožňuje např. zjistit, zda je `w1y` prvkem seznamu `[w1,w1x,w1y, w1z,w1f]`. Je třeba, abychom to věděli, protože s prvky seznamu budeme pracovat při formulaci pravidel. Vzory definujeme jako správné kombinace typů *W* a *V*. Při vytváření pravidel pak existence vzoru *vz* umožňuje kombinovat kmeny typu *W* s koncovkami typu *V*.
Chceme-li ovšem docílit toho, aby při splňování jednotlivých cílů popsaných pravidlem došlo k výběru prvku podvzor *W* ze seznamu *[WW]* a prvku podvzor V ze seznamu [VV], pak musíme tyto vztahy definovat.
PROLOG prohledává nejprve databázi a snaží se splnit první cíl, tj. dosadit za `vz(W,V)` nějaký `vz(w1,v1)`. Splnění dalšího cíle pak závisí na splnění cíle předchozího. Hledá se nějaké `km(Km,[WW])`, tedy kmen, kde se proměnná `Km` nahradí kmenem třeba `pán` a `[WW]` konkrétním seznamem podvzorů `[w1,w1x,w1y,w1z,w1f]`. Stejným způsobem se vyhledá i koncovka `t`. Proměnné `T,Cas,Num,[VV]` se nahradí `u,3,s,[v1,v2c]`. Při splňování těchto cílů musíme zajistit, aby první objekt predikátu vzor `w1` byl prvkem seznamu `[WW]` podvzorů kmene `Km` a aby druhý objekt predikátu vzor `v1` byl prvkem seznamu `[VV]` podvzorů koncovky `T`. Jak toho docílíme v PROLOGU?

Učebnice PROLOGU uvádějí (rekurzívní) definici predikátu member, která zní:

```
member(A,[A|T]) :- [A,A].
member(B,[_|T]) :-  member(B,T).
```

První řádek budeme číst takto:
`A` je prvkem seznamu, jestliže `A` lze srovnat s hlavou seznamu.

Druhý řádek budeme číst takto: `B` je prvkem seznamu, jestliže `B` lze srovnat s tělem seznamu.

Takto definujeme predikát member, tj. „být prvkem seznamu“. Jak jej nyní použijeme při formulování pravidel v PROLOGU? Pravidlo, kterým popisujeme konjunkci kmene a koncovky, existuje-li patřičný vzor, který takovou konjunkci dovoluje, vypadá s použitím predikátu member takto:

```
/*c-19*/       sb(sb(Km,Ts))  →          vz(W,V),
                                         km(Km,WW),member(W,WW),
                                         t(Ts,Cas,Num,VV),member(V,VV).
```

Slovně: Substantivum se skládá z kmene `Km` a z koncovky `T`, existuje-li vzor `vz(W,V)`, který umožňuje kombinovat kmen `Km` typu `[WW]`, pokud platí, že `W` je prvkem seznamu `[WW]` a koncovku `T` typu `[VV]`, pokud platí, že `V` je prvkem seznamu `[VV]`.

Slovesa

Mějme následující databázi, která bude obsahovat:
1) Propozice

a) o koncovkách sloves

b) o kořenech sloves

c) o kmenotvorných příponách sloves

d) o vzorech sloves

2) Pravidla
Pravidlo tvoření slovesných tvarů určitých a neurčitých, tj. kombinací (konjunkcí) složek kořen, kmenotvorná přípona, koncovka, pro které existuje společný vzor *vz*.

```
/*Pravidlo*/


/*c-20*/  v([Ko,Kp,Tv],Pers,Num,Mo)     →       vz(Wv,R,Vv),
                                                 ko(Ko,WX),
                                                 member(Wv,WX),
                                                 kp(Kp,R),
                                                 tv(Tv,Pers,Num,Mo,VX),
                                                 member(Vv,VX).

/*Koncovky*/


/*c-21*/  tv(tv(u),1,s,in,vv1)     →     [u].
          tv(tv('š'),2,s,in,vv2)   →     ['š'].

/*Kmenotvorné přípony*/
```

```
/*c-22*/  kp(kp(0),r1)                                  →      [0].
          kp(kp(e),r2)                                  →      [e].

/*Kořeny*/


/*c-23*/ ko(ko(nes),[wv1a,wv2a,wv3a,wv4a,wv5a])      →      [nes].
         ko(ko('nés'),[wv6a])                         →      ['nés'].

/*Vzory*/


/*c-24*/  vz(wv1,r1,vv1)            →      [wv1,r1,vv1].
          vz(wv1,r2,vv2)            →      [wv1,r2,vv2].
```

0.3.6 Složené tvary slovesné

Jistým přechodem mezi morfologií a syntaxí jsou složené tvary slovesné. Sestavení formálního popisu gramatiky složených slovesných tvarů je poměrně jednoduchým cvičením. Složitějším úkolem by bylo například zabudovat tento krátký úsek gramatiky do popisu české syntaxe. Je totiž značně obtížné zachytit všechny zákonitosti českého slovosledu, které umožňují vzájemnou izolaci jednotlivých složek složeného slovesného tvaru. Prostý popis složených slovesných tvarů v češtině je, jak již bylo řečeno, snadný.
Jaká data je třeba formálně popsat?
Složené tvary slovesné se, jak známo, skládají ze dvou složek:

1) Pomocné sloveso v určitém tvaru

2) Významové sloveso v neurčitém tvaru

3) Pomocné a významové sloveso se shodují v čísle, pokud jednotlivé složky vyjadřují tuto gramatickou kategorii.

4) Pokud se tvar skládá z neurčitých tvarů jak významového, tak pomocného slovesa, shodují se jednotlivé tvary také v rodě.

Databáze složených tvarů slovesných

Co bude zahrnovat databáze?

1) Seznamy tvarů pomocných sloves

2) Slovník neurčitých tvarů sloves významových

3) Pravidla tvoření složených slovesných tvarů

Poznámka:
Rozlišujeme dvě paradigmata slovesa *být*, a to jedno pro tvoření pasíva – *vb*, a druhé pro tvoření minulého času – *vbp*, kde se sloveso *být* ve třetí osobě sg. i pl. realizuje jako ∅.

```
/*DATABÁZE*/
/*SLOVESO být*/


/*c-25*/  vb(vb(jsem),1,s)            →          [jsem].
          vb(vb(jsi),2,s)             →          [jsi].
          vb(vb(je),3,s)              →          [je].
          vb(vb(jsme),1,p)            →          [jsme].
          vb(vb(jste),2,p)            →          [jste].
          vb(vb(jsou),3,p)            →          [jsou].

/*c-26*/  vbp(vbp(jsem),1,s)           →          [jsem].
          vbp(vbp(jsi),2,s)           →          [jsi].
          vbp(vbp(0),3,s)             →          [0].
          vbp(vbp(jsme),1,p)          →          [jsme].
          vbp(vbp(jste),2,p)          →          [jste].
          vbp(vbp(0),3,p)             →          [0].

/*c-27*/  vbf(vbf(budu),1,s)          →          [budu].
          vbf(vbf('budeš'),2,s)       →          ['budeš'].
          vbf(vbf(bude),3,s)          →          [bude].
          vbf(vbf(budeme),1,p)        →          [budeme].
          vbf(vbf(budete),2,p)        →          [budete].
          vbf(vbf(budou),3,p)         →          [budou].

/*c-28*/  vbk(vbk(bych),1,s)           →          [bych].
          vbk(vbk(bys),2,s)           →          [bys].
          vbk(vbk(by),3,s)            →          [by].
          vbk(vbk(bychom),1,p)        →          [bychom].
          vbk(vbk(byste),2,p)         →          [byste].
          vbk(vbk(by),3,p)            →          [by].

/*c-29*/  vbl(vbl(byl),mz,s)          →          [byl].
          vbl(vbl(byl),mn,s)          →          [byl].
          vbl(vbl(byla),f,s)          →          [byla].
          vbl(vbl(bylo),n,s)          →          [bylo].
          vbl(vbl(byli),mz,p)         →          [byli].
          vbl(vbl(byly),mn,p)         →          [byly].
          vbl(vbl(byly),f,p)          →          [byly].
          vbl(vbl(byla),n,p)          →          [byla].

/*c-30*/  vbll(vbll('býval'),mz,s)    →          ['býval'].
          vbll(vbll('býval'),mn,s)    →          ['býval'].
          vbll(vbll('bývala'),f,s)    →          ['bývala'].
          vbll(vbll('bývalo'),n,s)    →          ['bývalo'].
```

```
          vbll(vbll('bývali'),mz,p)     →          ['bývali'].
          vbll(vbll('bývaly'),mn,p)     →          ['bývaly'].
          vbll(vbll('bývaly'),f,p)      →          ['bývaly'].
          vbll(vbll('bývala'),n,p)      →          ['bývala'].

/*c-31*/  vl(vl(nesl),mz,s)             →          [nesl].
          vl(vl(nesl),mn,s)             →          [nesl].
          vl(vl(nesla),f,s)             →          [nesla].
          vl(vl(neslo),n,s)             →          [neslo].
          vl(vl(nesli),mz,p)            →          [nesli].
          vl(vl(nesly),mn,p)            →          [nesly].
          vl(vl(nesly),f,p)             →          [nes2ly].
          vl(vl(nesla),n,p)             →          [nesla].

/*c-32*/  vp(vp(nesen),mz,s)            →          [nesen].
          vp(vp(nesen),mn,s)            →          [nesen].
          vp(vp(nesena),f,s)            →          [nesena].
          vp(vp(neseno),n,s)            →          [neseno].
          vp(vp(neseni),mz,p)           →          [neseni].
          vp(vp(neseny),mn,p)           →          [neseny].
          vp(vp(neseny),f,p)            →          [neseny].
          vp(vp(nesena),n,p)            →          [nesena].
          vi(vi(pracovat))              →          [pracovat].
          vi(vi('pít'))                 →          ['pít'].
          vi(vi('stávkovat'))           →          ['stávkovat'].

/*c-33*/  vzm(vzm(Vl,Vbp))                  →      vl(Vl,Gen,Num),
                                                   vbp(Vbp,Pers,Num).

/*c-34*/  vzf(vzf(Vbf,Vi))                  →      vbf(Vbf,Pers,Num),
                                                   vi(Vi).

/*c-35*/  vzk(vzk(Vl,Vbk))                  →      vl(Vl,Gen,Num),
                                                   vbk(Vbk,Pers,Num).

/*c-36*/  vzkm(vzkm(Vbl,Vbk,Vl))            →      vbl(Vbl,Gen,Num),
                                                   vbk(Vbk,Pers,Num),
                                                   vl(Vl,Gen,Num).

/*c-37*/  vzp(vzp(Vb,Vp))                   →      vb(Vb,Pers,Num),
                                                   vp(Vp,Gen,Num).

/*c-38*/  vzpf(vzpf(Vbf,Vp))                →      vbf(Vbf,Pers,Num),
                                                   vp(Vp,Gen,Num).
```

```
/*c-39*/  vzpm(vzpm(Vbl,Vb,Vp))          →    vbl(Vbl,Gen,Num),
                                              vb(Vb,Pers,Num),
                                              vp(Vp,Gen,Num).

/*c-40*/  vzpk(vzpk(Vbl,Vbk,Vp))         →    vbl(Vbl,Gen,Num),
                                              vbk(Vbk,Pers,Num),
                                              vp(Vp,Gen,Num).

/*c-41*/  vzpkm(vzpkm(Vbl,Vbk,Vbll,Vp))  →    vbl(Vbl,Gen,Num),
                                              vbk(Vbk,Pers,Num),
                                              vbll(Vbll,Gen,Num),
                                              vp(Vp,Gen,Num).
```

0.4 Struktura české věty I

V současné lingvistice je obvyklé popisovat přirozené jazyky jako hierarchické systémy tvořené řadou subsystémů, které nazýváme roviny. Mluvíme pak o rovině fonologické, morfologické, syntaktické, sémantické a případně i pragmatické. Každá rovina je vymezena svými základními jednotkami a vztahy mezi nimi. Důležitou roli tu ovšem hrají i vztahy mezi jednotkami vzájemně sousedících rovin.

V tomto oddílu budeme věnovat pozornost rovině syntaktické i s jistým zřetelem k morfologii (oddíl 3), avšak jejich vzájemnými vztahy se budeme zabývat jen v míře nezbytné pro pochopení probírané problematiky.

Základní jednotkou syntaktické roviny je věta, kterou pro naše účely můžeme charakterizovat z několika hledisek:

„Věta je slovní vyjádření uzavřené myšlenky a základní jednotka jazykového sdělení uspořádaná a uzavřená po stránce mluvnické, významové i tvarové v celek (UčJČ, 1954)“.

„Věta představuje pole vztahů a větnou strukturu charakterizuje stupňovitá relační struktura. Strukturu odpovídající struktuře jistého typu vět (resp. její symbolický zápis) nazýváme též větným vzorcem (MČ 3, 1987, s. 8-9)“.

V syntaxi existují různé způsoby analýzy větné struktury. Pro českou (a slavistickou) gramatickou tradici je typická tzv. závislostní koncepce, která je založena na pojmu syntaktické závislosti. Jeho vymezení jsou rozličná, viz např. definici Šmilauerovu (NČS, 1966, s. 32), jež ovšem v prvé řadě vymezuje větný člen:

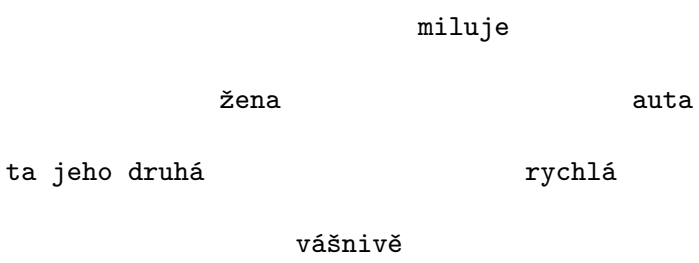
„Větnými členy jsou jen taková slova nebo skupiny slov, které mohou vytvářet skladební dvojice (skupiny)“.

GrK (1986, s. 201) praví: „Výrazy s funkcí nestejnorodých větných členů bývají spjaty vztahem, při kterém je jeden na druhém syntakticky závislý. Nazýváme ho vztah subordinační neboli subordinace (podřaďování). Sémanticky bývá výraz nadřazený (řídicí) výrazem podřazeným (závislým) blíže určován (determinován)“.

V MČ 3 (s. 15-14) se mluví o vztahu dominace, rozumí se jím však vztah syntaktické závislosti.

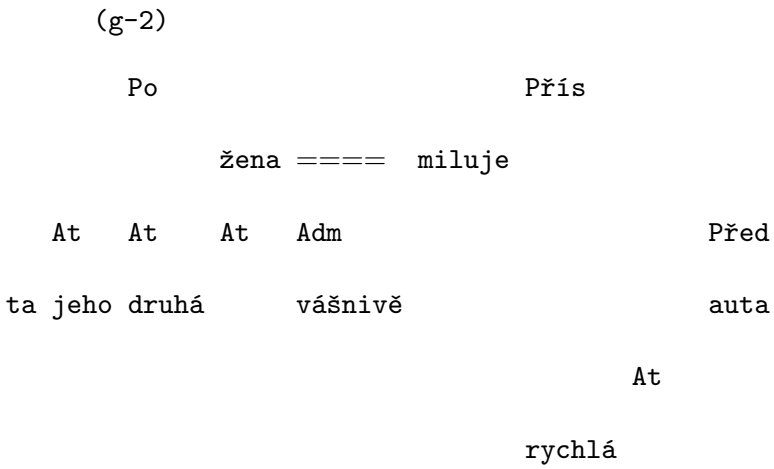
Zcela jednoduše řečeno, vztah syntaktické závislosti je vztah existující mezi větným prvkem řídícím a větným prvkem, který je mu podřízen. Analyzovat z tohoto hlediska strukturu věty pak znamená v předložené české větě najít všechny dvojice tvaru <řídicí prvek, závislý prvek>, tj. jinak řečeno, všechny skladebné dvojice.

Mějme větu
(v-1) *Ta jeho druhá žena vášnivě miluje rychlá auta.*
Najdeme v ní všechny skladebné dvojice a graficky (šipkami) znázorníme vztahy mezi řídícími a závislými prvky: (g-1)



V řadě klasických gramatik a ve školní praxi, jak už jsme naznačili, se pracuje i s větnými členy, které nejsou ničím jiným než jistou subklasifikací skladebných dvojic, přičemž jednotlivé větné členy jsou vymezeny svou schopností vázat na sebe jisté typy větných prvků, a tak odrážet jejich charakteristické a formálně signalizované funkce.

Rozšíříme-li analýzu věty o větné členy, bude mít graf (g-1) následující podobu:



V grafu (g-2) jsou tedy navíc proti (g-1) vyznačeny větné funkce jednotlivých prvků, tj. je v něm uvedeno, který prvek je kterým větným členem. Školský větný rozbor věty (v-1) by byl velmi podobný grafu (g-2), je však díky dosud užívanému způsobu vyznačování syntaktických vztahů (podtrháváním různými typy čar) málo názorný.

Nejčastěji se pracuje s následujícími větnými členy:

podmět – vstupuje do tzv. základní skladebné dvojice s přísudkem,

přísudek – vstupuje do základní skladebné dvojice s podmětem,

přívlastek – *shodný, neshodný*, závisí vždy na substantivu,

předmět – závisí na slovese nebo adjektivu,

příslovečné určení – závisí na slovese nebo adjektivu,

doplněk – má tzv. dvojí závislost, tj. závisí na substantivu i slovesu.

Větné členy, jak jsme si právě mohli povšimnout, jsou charakterizovány formálně, tj. jednak svou příslušností k určitému slovnímu druhu a jednak přítomností určitých gramatických kategorií, např. pádu, osoby, čísla aj. Mohou být a ve školní praxi často jsou charakterizovány i sémanticky, např. podle NČS (1966, s. 126) je „podmět ten člen predikační dvojice, jemuž se přísudkem přisuzuje nějaký znak“. „Přísudek (s.133) je pak ten člen predikační dvojice, jímž se podmětu přisuzuje nějaký znak“. U těchto definic si povšimněme nejen jisté tendence k cirkularitě, ale také toho, že jev formální povahy se v nich vymezuje též odkazem na své sémantické vlastnosti. Není jistě chybou, je-li jeden jev definován na základě dvou různých kritérií, problematické je však směšování formálních a sémantických kritérií, což lze pokládat za metodologický nedostatek, s nímž se však v klasických gramatikách setkáváme poměrně často.

Příklady analýzy (g-1) a (g-2) jsou velmi jednoduché, není ovšem těžké vidět, že naznačený způsob analýzy má pochopitelně obecnější platnost a je v literatuře (viz např. UčVR nebo

středoškolské učebnice) dobře propracován a aplikován na rozličné typy vět jednoduchých i souvětí.

V dalším bude naší snahou ukázat, že klasický (středoškolský) větný rozbor lze velmi dobře formalizovat s použitím formálních gramatik (zejména nekontextových). Transformujeme-li je pak do PROLOGU, dostáváme možnost experimentovat s větným rozborem na počítači.

0.5 Formální analýza české věty

Analýza české věty, jak byla nastíněna výše, není ryze formální, tj. není založena na čistě formálních rysech větných prvků, i když se o ně poměrně systematicky opírá. Položme si nyní otázku, zda je možné formulovat pravidla, resp. soubor pravidel, která by popisovala strukturu české (anglické, ruské, německé, francouzské, ...) věty tak, aby bylo možno automaticky (bez zásahu a přispění člověka) produkovat (generovat) české věty nebo je rozpoznávat (analyzovat).

Jinými slovy, zajímá nás, zda lze vytvořit soběstačný systém pravidel, který by se choval jako svého druhu automat a který by v jednom režimu české věty generoval (vytvářel) a v druhém je rozpoznával (analyzoval). Takový soběstačný systém vybudovat lze a může být tvořen souborem formálních pravidel, která mají jednoznačnou lingvistickou interpretaci. V dalším ukážeme, jak lze takový soubor formálních pravidel formulovat a jak jej můžeme poměrně jednoduše transformovat na počítačový program v programovacím jazyce PROLOG.

Pravidla, která se nyní pokusíme formulovat, nebudou však budována na vztahu syntaktické závislosti, nýbrž na vztahu tzv. bezprostředních složek. Řekneme, že dva (nebo více) větných prvků tvoří bezprostřední složky, jestliže patří k sobě a stojí bezprostředně vedle sebe. Např. ve spojení *velice chytrý student* jsou ve vztahu bezprostředních složek prvky <*velice, chytrý*> a prvky <*chytrý, student*>, nikoli však prvky <*velice, student*>. Je vidět, že rodilý mluvčí dovede na základě své znalosti jazyka poznat, které větné prvky k sobě patří a představují tedy bezprostřední složky.

Syntaktická analýza věty, při níž se využívá bezprostředních složek, je tedy založena na tzv. složkové koncepci. Složková koncepce analýzy věty je typická pro anglosaskou lingvistickou tradici (Wells, 1947, Chomsky, 1957) a v české lingvistice se objevuje až v posledních 15 – 20 letech v souvislosti s pokusy o formální popis české syntaxe (Sgall et al., 1967, Pala, 1982).

Poznamenejme ještě, že formální pravidla popisující strukturu české věty mohou mít jak tvar pravidel závislostních, tak i složkových, a platí, že oba způsoby analýzy jsou prakticky (i teoreticky) *ekvivalentní*. Složkové koncepci však dáváme v tomto textu přednost pro lepší propracovanost formálního aparátu (nekontextových gramatik), který je touto koncepcí využíván, a jeho notační blízkost programovacímu jazyku PROLOG.

0.5.1 Formální gramatiky

Soubor formálních pravidel, která umožňují generovat nebo rozpoznávat české věty a současně jim přiřazovat popisy jejich struktury, nazveme formální gramatikou (přesná definice následuje v dalším oddíle).

Vrátíme se nyní k větě

(v-1) *Ta jeho druhá žena vášnivě miluje rychlá auta.*
Ukázali jsme už, že tato věta má podmět a přísudek nebo, což je totéž, že ji lze rozčlenit na část podmětovou a část přísudkovou. Jestliže pro větu užijeme označení `S`, pro podmět `Np1` a pro přísudek `Vp`, pak tvrzení, že „věta (v-1) se skládá z podmětu a přísudku“ můžeme zapsat jako pravidlo:
(p-1) `S` → `Np1 Vp`,
Čtenář si právem může klást otázku, proč jsme nepoužili označení pomocí jiných symbolů, např. `V` pro větu, `Po` pro podmět a `Přís` pro přísudek a tedy i pravidla
(p-1a) `V` → `Po Přís`,
které by rovněž bylo správným zápisem našeho tvrzení.

Je pravda, že symboly lze volit různě, musí však být splněna jedna podmínka: vztahy mezi prvky věty musí být formulovány tak, aby výsledný popis adekvátně postihoval strukturu věty a byl ve shodě s naší lingvistickou intuicí.

V následujícím oddílu (Použitá symbolika) definujeme symboliku, která vychází z konvencí zavedených v současných gramatikách češtiny, a opíráme se přitom především o mezinárodní (latinskou) gramatickou terminologii.

Použitá symbolika

Jak si již čtenář mohl všimnout, při vytváření pravidel pracujeme s jistým souborem symbolů, které bývají charakterizovány jako syntaktické kategorie. Mohli jsme si již povšimnout, že v pravidlech se vyskytují symboly dvou druhů:

a) symboly označující slovní druhy, tj. v jistém smyslu základní větné prvky,

b) symboly označující větné složky, tj. jednotky o řád vyšší než slovní druhy.

Pro slovní druhy, jichž je podle většiny českých gramatik devět, a jejich podtřídy užíváme následujícího označení:

- `N1` – podstatné jméno (substantivum, nomen) v nominativu (`= 1`),
  `N2` v genitivu, ...
- `Npr1` – substantivum – vlastní jméno v nominativu, `Npr2` v genitivu, ...
- `Pnd1` – ukazovací zájmeno (pronomen demonstrativní) v nominativu,
  `Pnd2` v genitivu, ...
- `Pos1` – přivlastňovací zájmeno (posesívní pronomen) v nominativu, `Pos2, ...`
- `Per1` – osobní (personální) zájmeno v nominativu, `Per2, ...`
- `Pr1` – vztažné (relativní) zájmeno v nominativu, `Pr2, ...`
- `Pun1` – neurčité zájmeno v nominativu, `Pun2, ...`
- `Pq1` – tázací zájmeno v nominativu, `Pq2, ...`

- `Numk1` – číslovka základní (numerale kardinální) v nominativu, `Numk2, ...`
- `Numo1` – číslovka řadová (numerale ordinální) v nominativu, `Numo2, ...`
- `A1` – přídavné jméno (adjektivum) v nominativu, `A2, ...`
- `V1, V2, V3` – finitní tvar slovesa v 1., 2., 3. osobě sg. préz. ind., ...
- `Vr1, Vr2, Vr3` – finitní tvar slovesa zvratného (reflexíva) v 1., 2., 3. osobě sg. préz. ind., ...
- `Vm1, Vm2, Vm3` – finitní tvar slovesa modálního v 1., 2., 3. osobě sg. préz. ind., ...
- `Vf1, Vf2, Vf3` – finitní tvar slovesa fázového v ,1., 2., 3. osobě sg. préz. ind., ...
- `Vpas` – příčestí trpné (participium pasívní)
- `Vl` – příčestí minulé (l-ové participium)
- `Vi` – neurčitý způsob slovesa (infinitiv)
- `Vb` – finitní (určitý) tvar slovesa *být*
- `Vbl` – příčestí minulé slovesa *být*
- `Vbc` – kondicionálový tvar slovesa *být*
- `Vml` – příčestí minulé modálního slovesa
- `Vfl` – příčestí minulé fázového slovesa
- `Ad` – příslovce obecně (adverbiale)
- `Adm` – příslovce způsobu (adverbiale modi)
- `Adq` – příslovce způsobu, a to míry
- `Adl` – příslovce místa (adverbiale loci)
- `Adt` – příslovce času (adverbiale temporis)
- `Prep` – předložka (prepozice)
- `K` – spojka (konjunkce), `Ks` – subordinační, `Kk` – koordinační
- `Prt` – částice (partikule)
- `R` – částice zvratná (reflexívní)
- `In` – citoslovce (interjekce)
- `Fm` – symbol pro čárku, tečku, otazník nebo vykřičník

Uvnitř jednotlivých slovních druhů se mohou vyskytovat i dosti složité subklasifikace, které jsou zde uvedeny v základních obrysech, např. u zájmen, sloves a adverbií. Není však obtížné je v případě potřeby dále doplnit. U větných složek, které do jisté míry odpovídají větným členům uváděným v českých gramatikách, rozlišujeme aspoň tyto:

- `S` – věta (sentence), nejvyšší větná složka
- `Np1, 2, 3, ...` – jmenná skupina (nominální fráze) v nominativu, genitivu, ..., tj. větná složka, jejímž konstitutivním prvkem je substantivum
- `Pnp1, 2, ...` – jmenná skupina pronominální, jejím konstitutivním prvkem je příslušný typ zájmena, např. osobní, neurčité, tázací, vztažné apod.
- `Pp2, 3, ...` – předložková jmenná skupina v genitivu, dativu, ... Od jmenné skupiny se liší jen tím, že navíc obsahuje předložku v příslušném pádě
- `Vp` – slovesná skupina, jejím konstitutivním prvkem je příslušný finitní slovesný tvar (jednoduchý nebo složený)
- `Vpb` – slovesná skupina, jejím konstitutivním prvkem je sloveso *být* ve funkci spony
- `Adp` – adverbiální skupina, jejím konstitutivním prvkem je vhodný typ adverbia
- `Ap` – adjektivní skupina, jejím konstitutivním prvkem je adjektivum.

Uvedený výčet není pochopitelně vyčerpávající, obsahuje však hlavní větné složky, s nimiž je potřeba při formálním popisu české věty počítat.

V češtině se přirozeně neobejdeme bez symbolů pro gramatické významy, jako jsou *pád, číslo, rod, osoba, čas, způsob* a další. Pozorný čtenář si již jistě povšiml, že jednotlivé pády označujeme čísly `1, 2, ..., 7`, ale pro úplnost uvedeme všechny symboly, které připadají v úvahu:

- pády: `1` – nominativ, `2` – genitiv, `3` – dativ, `4` – akuzativ, `5` – vokativ,`6` – lokál, `7` – instrumentál
- číslo: `sg` – jednotné číslo (singulár), `pl` – množné číslo (plurál)
- rod jmenný: `mz` – rod mužský životný, `mn` – mužský neživotný, `f` – ženský, `n` – střední
- osoba: `1` – první osoba, `2` – druhá osoba, `3` – třetí osoba
- gramatický čas: `préz` – přítomný čas (prézens), `pret` – minulý čas (préteritum), `fut` – budoucí čas (futurum)
- slovesný způsob: `ind` – oznamovací způsob (indikativ), `imp` – rozkazovací způsob (imperativ), `c` – podmiňovací (kondicionál)
- rod slovesný: `act` – aktívní (aktívum), `pas` – pasívní (pasívum)
- vid slovesný: `impf` – nedokonavý (imperfektivní), `pf` – dokonavý (perfektivní).

Způsob kombinování symbolů označujících gramatické významy se symboly pro slovní druhy a větné složky jsme již naznačili, viz symboly jako `Np1, V3, V1` apod.

0.5.2 Formální gramatika – pokračování

Můžeme nyní pokračovat ve formulování pravidel naší gramatiky. Formulovali jsme již pravidlo (p-1). Na jeho levé straně se jako první objevuje symbol `Np1`, kterého užíváme pro označení podmětové části (subjektu) věty. Jak jej chápat, vysvítá z pravidla:
(p-2) `Np1` → `Pnd1 Pos1 Num1 N1`,
které říká, že podmětová část věty (označená symbolem `Np1` na levé straně pravidla (p-2)) je zde tvořena ukazovacím zájmenem v nominativu (`Pnd1`), přivlastňovacím zájmenem v nominativu (`Pos1`), číslovkou řadovou (`Num1`) v nominativu a substantivem v nominativu (`N1`). Jak patrno, pravidlo (p-2) postihuje slovnědruhovou strukturu podmětové části věty a její definující formální příznak – `nominativ`.

Následuje specifikace druhého symbolu z pravé strany pravidla (p-1), jímž je `Vp` označující přísudkovou část věty (predikát):
(p-3) `Vp` → `Adgm V3 Np4`
Přísudková část věty se v tomto případě, jak patrno, skládá ze tří složek, a to z příslovečného určení způsobu (`Adgm`), z určitého tvaru slovesného ve 3. os. sg. ind. préz. (`V3`) a předmětu (`Np4`). Pravidlo (p-3) obsahuje na pravé straně složky větněčlenské povahy (`Adgm, Np4`) a složku povahy slovnědruhové (`V3`).

Další pravidlo charakterizuje příslovečné určení způsobu (`Adgm`) uvedené na pravé straně předchozího pravidla:
(p-4) `Adgm` → `Adm`
Složka `Adgm` je na slovnědruhové úrovni vyjádřena příslovcem způsobu.
Strukturu předmětu (objektu) označeného `Np4` lze zachytit např. takto:
(p-5) `Np4` → `A4 N4`,
což znamená, že předmět (`Np4`) je tvořen jmennou skupinou obsahující adjektivum v akuzativu (`A4`) a substantivum v akuzativu (`N4`).

Nyní následují pravidla, která bychom mohli charakterizovat jako slovníková, – jejich prostřednictvím jednotlivým již uvedeným slovním druhům přiřadíme konkrétní slovní tvary (vyjadřující i konkrétní gramatické kategorie). Tato pravidla tedy definují slovník, který bude sloužit ke generování nebo rozpoznávání konkrétních českých vět, např. v našem jednoduchém případě věty jako (v-1).

(p-6)	Pnd1	→	{ta, tato,...}
(p-7)	Pos1	→	{moje, tvoje, jeho,...}
(p-8)	Num1	→	{první, druhá,...}
(p-9)	N1	→	{žena, manželka,...}
(p-10)	V3	→	{miluje, zbožňuje,...}
(p-11)	A4	→	{rychlá, silná,...}
(p-12)	N4	→	{auta, vozidla,...}
(p-13)	Adm	→	{vášnivě,divoce,...}

Svorky v pravidlech (p-6)–(p-13) je třeba chápat jako vyznačující množiny (množinové), pravé strany pravidel mohou tedy obsahovat i dosti velký (avšak vždy konečný) počet slovních tvarů, např. substantiv v nominativu nebo akuzativu apod. Při použití pravidla lze však v daném okamžiku vybrat z pravé strany vždy jen jeden slovní tvar.

Pravidla (p-1)–(p-13) představují konečnou množinu formálních pravidel, kterou budeme nazývat formální gramatikou (přesnou definici viz v následujícím oddíle). Pravidla naší gramatiky – označme ji **g1** – se aplikují tak, že levá strana pravidla se nahradí odpovídající pravou stranou (operátor → čteme „nahraď“). Symboly vyskytující se jak na levých, tak i na pravých stranách pravidel nazveme neterminálními, zatímco symboly, které se objevují jen na pravých stranách našich pravidel, charakterizujeme jako terminální. Postupným aplikováním pravidel dostaneme následující posloupnost řádků:

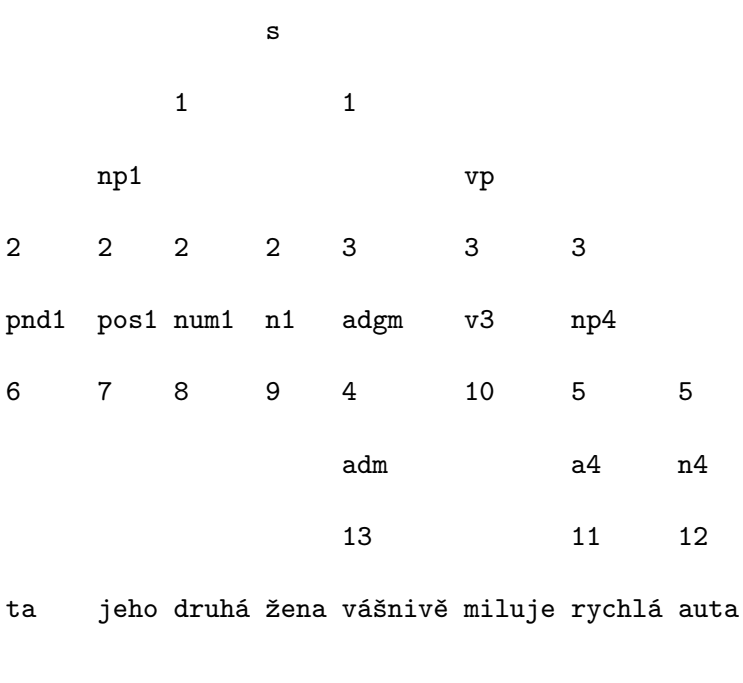
(g-3)

```
S
Np1 Vp (1)
Pnd1 Pos1 Num1 N1 Vp (2)
ta Pos1 Num1 N1 Vp (6)
ta jeho Num1 N1 Vp (7)
ta jeho druhá N1 Vp (8)
ta jeho druhá žena Vp (9)
ta jeho druhá žena Adgm V3 Np4 (3)
ta jeho druhá žena Adm V3 Np4 (4)
ta jeho druhá žena vášnivě V3 Np4 (13)
ta jeho druhá žena vášnivě miluje Np4 (10)
ta jeho druhá žena vášnivě miluje A4 N4 (5)
ta jeho druhá žena vášnivě miluje rychlá N4 (11)
ta jeho druhá žena vášnivě miluje rychlá auta (12)
```

Tato posloupnost řádků se nazývá derivace a zachycuje odvození věty (v-1) v gramatice **g1**. V závorkách napravo je vždy uvedeno číslo použitého pravidla. Jak patrno ze způsobu nahrazování neterminálních symbolů, nahrazuje se vždy nejlevější symbol, jde tedy o tzv. levou derivaci.

Snadno lze vidět, že derivace (g-3) definuje graf-strom, tj. strukturní popis věty (v-1), který má následující tvar (hrany grafu jsou ohodnoceny čísly použitých pravidel stejně jako v derivaci (g-3)):

(g-4)



Graf-strom (g-4) věty (v-1) vyhovuje plně našim požadavkům, neboť obsahuje

1. strukturní popis věty v podobě stromové struktury,
2. samotnou větu (v-1) – na posledním řádku.

Můžeme tedy říci, že soubor pravidel označený jako **g1** jednak generuje strukturní popis věty (v-1) a jednak jej spojuje s větou (v-1) jako takovou.

0.5.3 Definice gramatiky

Intuitivní vymezení gramatiky **g1** nyní zpřesníme a uvedeme její formální definici v podstatě tak, jak je uvedena v práci Češky a Rábové (1985). Všechny potřebné pojmy jsou definovány tak, že jim lze porozumět na základě znalosti středoškolské matematiky.

Doporučujeme čtenáři, aby si tuto partii pozorně prostudoval, neboť z hlediska celkového porozumění formálnímu přístupu k popisu přirozeného jazyka má klíčovou povahu. Obsahuje totiž přesnou formální definici pojmu gramatiky a také klasifikaci gramatik. Je důležité uvědomit si, že tento přístup je neutrální vzhledem ke kterémukoli přirozenému jazyku, což znamená, že je také bezprostředně aplikovatelný nejen v bohemistice, ale i anglistice, germanistice atd. Uvádíme ji zde i proto, že tento přístup k teorii gramatik se v běžných na fakultě užívaných učebnicích prakticky (až na některé výjimky) nevyskytuje.

Lingvisticky orientovaný výklad uvedené problematiky je v klasické podobě podán u Chomského (1966), což je práce, kterou by si měl přečíst každý adept lingvistiky. Čtenáři, který se chce dovědět více o formální teorii jazyků a gramatik a vztazích k teorii automatů, doporučujeme věnovat pozornost např. práci Novotného (1988) a také kapitolám Chomského a Millera z knihy *Handbook of Mathematical Psychology* (Chomsky, Miller, 1963).

Gramatika v tomto chápání představuje formální prostředek, pomocí něhož můžeme vymezit jak konečné tak nekonečné jazyky, přičemž gramatika sama je konečná.

Nejprve uvedeme potřebné výchozí pojmy: Prvním z nich je abeceda, jíž rozumíme neprázdnou množinu prvků – symbolů abecedy. Jako příklad lze uvést třeba latinskou abecedu čítající 52 symbolů (velká i malá písmena) nebo českou abecedu, která celkem obsahuje 82 symbolů.

Dalším je řetězec (ev. slovo). Řetězcem nad danou abecedou rozumíme nějakou posloupnost symbolů abecedy. Posloupnost, která neobsahuje žádný symbol, nazveme prázdným řetězcem a budeme ji značit e.

Přesněji řečeno, řetězec nad abecedou T definujeme takto:

1. prázdný řetězec e je řetězec nad abecedou T,
2. je-li x řetězec nad T a $a \in T$, pak xa je řetězec nad T,
3. y je řetězec nad T tehdy a jen tehdy, lze-li y získat aplikací pravidel (1) a (2).

Máme-li řetězce x a y a připojíme-li y za x, vznikne řetězec xy. Této operaci říkáme zřetězení (konkatenace).

Je dána abeceda T. Pak T^* je množina všech řetězců nad abecedou T včetně prázdného řetězce a T^+ je množina všech řetězců nad T kromě prázdného řetězce e, tj. $T^* = T^+ \cup \{e\}$. Množinu L, pro niž platí $L \subseteq T^*$ (případně $L \subseteq T^+$, pokud $e \neq L$), nazýváme jazykem nad abecedou T. Jazykem tedy může být libovolná podmnožina řetězců nad danou abecedou.

Budeme pracovat se dvěma disjunktními abecedami (množinami) symbolů:

1. abecedou N (množiny) neterminálních symbolů, které v popisu jazyka interpretujeme jako syntaktické kategorie,
2. abecedou T (množiny) terminálních symbolů, jež interpretujeme (nejčastěji) jako slova daného jazyka,
3. sjednocení obou abeced N a T, tj. $N \cup T$, nazýváme slovníkem gramatiky.

V dalším výkladu budeme pro zápis terminálních a neterminálních symbolů a z nich tvořených řetězců užívat následující konvence, jíž jsme se ostatně přidržovali již výše:

1. $a, b, c, d, ...$ – označují terminální symboly
2. $A, B, C, D, ...$ – označují neterminální symboly
3. $U, V, ..., Z$ – označují terminální nebo neterminální symboly
4. $\alpha, \beta, ..., \omega$ – označují řetězce terminálních a neterminálních symbolů

5. $u, v, ..., z$ – označují řetězce pouze terminálních symbolů

Nyní jsme připraveni definovat formální gramatiku $G1$.

Gramatika $G1$ je uspořádaná čtveřice

$g1 = \{N, T, P, S\}$,

- kde N je konečná množina neterminálních symbolů, které interpretujeme jako syntaktické kategorie,
- T je množina terminálních symbolů, jež interpretujeme jako konkrétní české slovní tvary, a platí, že $N \cap T = \emptyset$,
- P je konečná podmnožina kartézského součinu $(N \cup T)^* N \quad (N \cup T)^*$ x $(N \cup T)^*$,
- $S \in N$ je tzv. vyznačený počáteční symbol gramatiky G,
- prvek (α, β) množiny P nazýváme přepisovacím pravidlem a budeme jej zapisovat ve tvaru $\alpha \rightarrow \beta$. Řetězec α nazýváme levou stranou pravidla, řetězec β pravou stranou přepisovacího pravidla.

Jádrem gramatiky tedy je konečná množina přepisovacích pravidel. Každé pravidlo má tvar uspořádané dvojice (α, β) řetězců a stanovuje možné nahrazení řetězce α řetězcem β. Řetězec α obsahuje alespoň jeden neterminální symbol, řetězec β je prvek sjednocení $(N \cup T^*)$.

Nechť λ a μ jsou řetězce z $(N \cup T)^*$. Pak mezi nimi platí relace $\overset{G}{\Longrightarrow}$, která se nazývá přímá derivace, jestliže řetězce λ a μ můžeme zapsat ve tvaru

$$\lambda = \gamma\alpha\delta$$
$$\mu = \gamma\beta\delta,$$

kde γ a δ jsou libovolné řetězce z $(N \cup T)^*$ a $\alpha \rightarrow \beta$ je nějaké přepisovací pravidlo.

Dojdeme-li v posloupnosti přímých derivací k řetězci, který obsahuje pouze terminální symboly, pak již nelze aplikovat žádné přepisovací pravidlo a proces generování končí. Z této skutečnosti, která plyne z definice pravidla, je odvozen název množiny T jako množiny terminálních symbolů.

Jestliže existuje posloupnost přímých derivací $\nu_{i-1} \Longrightarrow \nu_i, i = 1, ..., n, n > 1$ taková, že platí: $\lambda = \nu_0 \Longrightarrow \nu_1 \Longrightarrow ... \Longrightarrow \nu_{n-1} \Longrightarrow \nu = \mu$, nazýváme ji derivace a značíme ji $\overset{+}{\Longrightarrow}$. Tuto posloupnost nazýváme derivací délky n.

Jestliže v gramatice G platí pro řetězce λ a μ relace $\lambda \overset{+}{\Longrightarrow} \mu$ nebo identita $\lambda = \mu$, pak píšeme $\lambda \overset{*}{\Longrightarrow} \mu$. Relace $\overset{*}{\Longrightarrow}$ je tranzitivním a reflexívním uzávěrem relace přímé derivace.

Důležitým prostředkem pro grafické vyjádření struktury věty (její derivace) je graf-strom, který se nazývá derivační nebo syntaktický strom věty. Přesněji řečeno, strom je orientovaný acyklický graf s následujícími vlastnostmi:

1. existuje jediný uzel, tzv. kořen stromu, do něhož nevstupuje žádná hrana,
2. do všech ostatních uzlů vstupuje právě jedna hrana,
3. uzly, z nich žádná hrana nevystupuje, se nazývají koncové (terminální) nebo také listy,
4. při kreslení se zachovává konvence, že kořen je nejvýše a všechny hrany jsou orientovány směrem dolů,
5. uspořádání hran zachovává slovoslednou relaci, tj. pořadí slov ve větě (zleva doprava).

Je-li G gramatika, pak řetězec $\alpha \in (N \cup T)^*$ se nazývá větná forma právě tehdy, když platí $S \stackrel{*}{\Longrightarrow} \alpha$, tj. řetězec α je generovatelný z počátečního symbolu S. Větná forma, která obsahuje pouze terminální symboly, se nazývá věta. Jazyk $L(G)$ generovaný gramatikou G je definován množinou všech vět:
$L(G) = \{w | S \stackrel{*}{\Longrightarrow} w \wedge w \in T^*\}$.

Množinu vět generovaných gramatikou nazýváme jazyk a dále rozlišujeme slabou generativní kapacitu gramatiky, jíž je jazyk $L(G)$ (množina všech vět generovaných gramatikou G), který je gramatika G schopna generovat, a silnou generativní kapacitu – což je množina syntaktických stromů (strukturních popisů) přiřazovaných větám jazyka L generovaného gramatikou G.

0.5.4 Typy gramatik

Gramatiky lze klasifikovat do typů podle tvaru přepisovacích pravidel. Je obvyklé vymezovat čtyři typy gramatik, které se nazývají typ 0, typ 1, typ 2 a typ 3.

Typ 0

Gramatika typu 0 obsahuje pravidla v nejobecnějším tvaru, kdy platí
$\alpha \to \beta, \alpha \in (N \cup T)^* N \quad (N \cup T)^*, \beta \in (N \cup T)^*$.
Protože se neklade žádné omezení na tvar pravidel a povoluje se přepisovat řetězce na řetězce, mluvíme také o neomezených přepisovacích systémech.

Typ 1

Gramatika typu 1 obsahuje pravidla tvaru
$\alpha A \beta \to \alpha \gamma \beta, A \in N, \alpha, \beta \in (N \cup T)^*, \gamma \in (N \cup T)^+$ nebo $S \to e$.
Gramatiky typu 1 se také nazývají gramatikami kontextovými, protože v kontextových pravidlech lze neterminální symbol A nahradit řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α.

Kontextové gramatiky neobsahují pravidla tvaru $\alpha A \beta \to \alpha \beta$, a tedy nepřipouštějí, aby neterminální symbol byl nahrazen prázdným řetězcem. Jinými slovy, při generování věty nemůže dojít ke zkracování generovaných řetězců.

Typ 2

Gramatika typu 2 obsahuje pravidla tvaru
$A \rightarrow \gamma, A \in N, \gamma \in (N \cup T)^*$.
Nazýváme je také gramatikami nekontextovými, protože nahrazení neterminálního symbolu A na levé straně pravidla řetězcem γ lze provést bez ohledu na jakékoli okolí, v němž by se neterminální symbol A mohl vyskytovat.

Pro popis syntaktické stavby přirozených jazyků jsou nejzajímavější právě nekontextové gramatiky. Gramatika **g1** popsaná výše je příkladem nekontextové gramatiky pro češtinu. Podobně gramatiky vymezených klauzulí v PROLOGU, o nichž bude řeč níže, vycházejí z formalismu nekontextových gramatik.

Typ 3

Gramatika typu 3 je tvořena pravidly ve tvaru
$A \rightarrow xB$ nebo $A \rightarrow x; A, B \in N, x \in T^*$.
Protože jediný možný neterminální symbol na pravé straně pravidla stojí zcela vpravo, mluvíme také o pravé lineární gramatice. Poznamenejme ještě, že gramatiky typu 3 se také nazývají regulárními gramatikami.

Pro práci s přirozenými jazyky, jak jsme prakticky ukázali výše, zůstávají východiskem gramatiky nekontextové. V lingvistické literatuře posledních 20-30 let se sice spotřebovalo mnoho papíru na argumenty, které si kladly za cíl ukázat, že nekontextové gramatiky jsou pro popis přirozených jazyků nedostačující a že je potřeba zavést gramatiky silnější – transformační (viz již Chomsky, 1957), poslední práce (např. Gazdar, 1982, Gazdar, Mellish, 1989, Pereira, 1983) však obsahují jejich určitou rehabilitaci. Zejména se podařilo ukázat, že implementace nekontextových gramatik v PROLOGU v podobě tzv. gramatik vymezených klauzulí (definite clause grammars = DCG), o nichž bude vzápětí řeč, umožňuje zachovat nekontextovou podobu pravidel a současně získat kontextovou citlivost tak potřebnou pro formální popis gramatické shody a dalších kontextově podmíněných gramatických jevů v přirozených jazycích.

0.6 Gramatiky v PROLOGU

Nyní je naším cílem přepsat výše uvedenou gramatiku **g1** tak, aby s ní bylo možno pracovat jako s gramatikou v PROLOGU. Nekontextovým gramatikám, jako je **g1**, v PROLOGU odpovídají gramatiky vymezených klauzulí – DC gramatiky.

0.6.1 Nekontextové gramatiky a DC gramatiky

Gramatická pravidla DC gramatiky jsou velmi podobná pravidlům **g1**, mají stejně jako ona levou a pravou stranu a operátor $\rightarrow$. Podstatný rozdíl je však v tom, že jednotlivé neterminální symboly v **g1** musí být v DC gramatice zapsány jako predikáty s příslušným počtem argumentů.

Nekontextovou gramatiku **g1** přepíšeme tedy jako DC gramatiku se jménem `g1.pl`, tj. jako textový soubor s tímto jménem. Soubor vytvoříme pomocí některého z již zmíněných editorů,

nejlépe NE nebo CSED. S výhodou lze též použít editoru vestavěného v ARITY PROLOGU v. 5.1.

Poznámka:
Nedoporučujeme používat WP 5.1 ani T602, protože oba tyto editory vkládají na začátek svých souborů řadu znaků, které jsou pro PROLOG nečitelné. Je sice možno použít funkce EXPORT, pomocí níž lze naše soubory konvertovat na čisté ASCII soubory, se kterými si již PROLOG poradí. Celkově však jde o netriviální proceduru, takže za jednodušší (zejména pro začátečníky) pokládáme použití výše uvedených editorů nezpůsobujících žádné komplikace.

Při přepisování budeme dodržovat tyto konvence:

1. výraz označující konstantu v PROLOGU musí začínat malým písmenem,
2. výraz označující proměnnou musí začínat velkým písmenem,
3. za každým pravidlem píšeme tečku,
4. `/* tento text */` jsou pro PROLOG závorky, do nichž umisťujeme poznámky nebo údaje, které potřebujeme jen my sami, a PROLOG je ignoruje. To se týká např. číslování pravidel gramatiky nebo hlaviček oddělujících vlastní pravidla gramatiky od pravidel definujících slovník (viz níže).

Poznamenáváme, že očíslování pravidel v nekontextové gramatice **g1** a v DC gramatice `g1.pl` je shodné, takže čtenář může porovnávat snadno podobu pravidel v **g1** a v `g1.pl`. Princip přepisu pravidel z nekontextové gramatiky do DC gramatiky je následující:

Vyjděme z pravidla gramatiky **g1**
(p-1) `S` → `Np1 Vp`,
jež, jak víme, rozkládá větu na jmennou skupinu v nominativu a slovesnou skupinu, což je vyjádřeno příslušnými neterminálními symboly. V DC gramatice nemůžeme použít jednoduchých neterminálních symbolů jako v **g1**, ale musíme je nahradit příslušnými predikáty. Místo `S` budeme mít v `g1.pl` predikát `s(s(Np1,Vp))`, který má tři argumenty: z nichž dva jsou pro nás nedostupné a také v rámci DC gramatiky neviditelné a jeden – `s(Np1,Vp)` – zajišťuje vytvoření podstromu definovaného pravidlem (p-1) v grafu-stromu generované nebo rozpoznávané věty – (v-1). Predikát (neterminál) `s` je splněn, jsou-li splněny predikáty odpovídající neterminálům na pravé straně pravidla (p-1):
`NP1` tedy odpovídá `np1(Np1)` a `VP` odpovídá `vp(Vp)`, takže (p-1) odpovídá
`/*p-1*/ s(s(Np1,Vp))` → `np1(Np1),vp(Vp).`
Predikáty `np1` a `vp1` jsou stejně jako predikát `s` tříargumentové. Podobně budeme postupovat i u dalších pravidel gramatiky **g1**.

Nyní již můžeme uvést přepis pravidel **g1** do pravidel DC gramatiky:

```
/* gramatika g1.pl */


/*p-1*/     s(s(Np1,Vp))          →     np1(Np1), vp(Vp).
/*p-2*/     np1(np1(N1))          →     n1(N1).
/*p-2a*/    np1(np1(Pnd1,N1))     →     pnd1(Pnd1), n1(N1).
```

```
/*p-2b*/   np1(np1(A1,Np1))       →    a1(A1), np1(Np1).
/*p-2c*/   np1(np1,(Pos1,N1))     →    pos1(Pos1),n1(N1).
/*p-2d*/   np1(np1,(Num1,N1))     →    num1(Num1),n1(N1).
/*p-3*/    vp(vp(Adgm,V3,Np4))    →    adgm(Adgm),v3(V3),np4(Np4).
/*p-3a*/   vp(vp(V3,Np4))         →    v3(V3), np4(Np4).
/*p-3b*/   vp(vp(Adgm,V3))        →    adgm(Adgm),v3(V3).
/*p-3c*/   vp(vp(V3))             →    v3(V3).
/*p-4*/    adgm(adgm(Adm))        →    adm(Adm).
/*p-5*/    np4(np4(A4,N4))        →    a4(A4),n4(N4).


/* slovník */


/*p-6*/    pnd1(pnd1(ta))         →    [ta].
/*p-7*/    pos1(pos1(jeho))       →    [jeho].
           pos1(pos1(moje))       →    [moje].
/*p-8*/    num1(num1(první))      →    [první].
           num1(num1(druhá))      →    [druhá].
/*p-9*/    n1(n1(žena))           →    [žena].
           n1(n1(babička))        →    [babička].
/*p-10*/   v3(v3(miluje))         →    [miluje].
           v3(v3(nenávidí))       →    [nenávidí].
/*p-11*/   a1(a1(krásná))         →    [krásná].
           a1(a1(chytrá))         →    [chytrá].
/*p-12*/   a4(a4(rychlá))         →    [rychlá].
           a4(a4(silná))          →    [silná].
/*p-13*/   n4(n4(auta))           →    [auta].
           n4(n4(kuřata))         →    [kuřata].
/*p-14*/   adm(adm(vášnivě))      →    [vášnivě].
           adm(adm(bláznivě))     →    [bláznivě].
```

Čtenář si jistě povšimne, že proti **g1** obsahuje `g1.pl` několik pravidel navíc. Jejich užití lze snadno vyzkoušet, a tak si ověřit, v čem rozšiřují výchozí nekontextovou gramatiku **g1**. V cvičeních, která následují, je podrobněji naznačen efekt způsobený např. pravidlem `/*p-2b*/`, které má tu vlastnost, že neterminální symbol (predikát) `np1` se v něm vyskytuje na levé i pravé straně. Takové pravidlo se nazývá rekurzívní. Jsou to právě rekurzívní pravidla, která umožňují postihnout opakující se pravidelnosti syntaktických struktur přirozeného jazyka a díky nimž, jak se čtenář může snadno sám přesvědčit, může konečná gramatika (tj. gramatika s konečným počtem pravidel) generovat nekonečně mnoho vět.

0.6.2 Práce s DC gramatikou na počítači

Jednotlivé řádky, které jsou v dalším psány `strojopisným typem písma`, jsou často přímo příkazy, které píšeme na klávesnici svého počítače. Pracujeme-li na počítači typu IBM PC pod

operačním systémem MS-DOS, přejdeme nejprve do adresáře, v němž se nalézá náš PROLOG. Nacházíme-li se na disku `C:`, bude na obrazovce pravděpodobně prompt:
`C:\>`,
na který napíšeme
`cd \api`,
což je příkaz (`change directory` – `změň adresář`) pro přechod (zde) z adresáře `C:\` do adresáře `C:\api`, v němž je uložen náš systém PROLOG.

Poznámka:
Pro toto vydání předpokládáme, že nejčastěji užívaným systémem PROLOGU bude ARITY PROLOG, a to buď v. 4.2, nebo v. 5.1. Tímto faktem je dána volba některých příkazů a jejich součástí, např. `api`, nebo extenze souborů `ari`. Je také nutno upozornit na rozdíl mezi verzemi ARITY PROLOGU: ve starší v. 4.2 nelze pracovat s jednotlivými menu (jídelníčky), a ne vždy je k dispozici vestavěný editor, zatímco nová v. 5.1 poskytuje uživateli již větší komfort. Nebudeme zde detailně popisovat rozdíly mezi jednotlivými verzemi a předpokládáme, že uživatel se s oběma verzemi seznámí sám a bude v lepším případě pracovat s tou, která mu bude lépe vyhovovat, – v horším případě s tou, kterou bude mít k dispozici.

Jakmile máme na obrazovce prompt:
`C:\api>`
můžeme již spustit samotný PROLOG. To učiníme nejspíše příkazem
`api`,
který po stisknutí příkazové klávesy `ENTER` vyvolá hlavičku ARITY PROLOGU nebo u v. 5.1 jeho menu. Poté se ihned objeví typický prompt PROLOGU
`?-` ,
který nás informuje jednak o tom, že jsme uvnitř systému PROLOG, a také o tom, že PROLOG je připraven přijímat naše příkazy ve tvaru predikátů.

V ARITY PROLOGU je preprocesor pro DC gramatiky zabudován již uvnitř systému, takže nemusíme používat speciálního predikátu `consultg` (tj. preprocesoru), který u jednodušších verzí PROLOGU slouží k překladu DC gramatik do vlastního systému PROLOGU.

Pokud se v souboru `g1.pl` při zavádění odhalila nějaká syntaktická chyba (chybí závorky, tečky ap.), PROLOG nám to oznámí chybovým hlášením.

Např. jsme při psaní pravidel gramatiky `g1.pl` zapomněli napsat za pravidlem `/*p-2*/` tečku. Tuto chybu ARITY PROLOG ohlásí tak, že na místo, kde v souboru `g1.pl` chybí tečka, ukáže na obrazovce svislou šipkou a současně vypíše hlášení „`missing operator`“, což česky znamená „`chybějící operátor`“ (jímž je ona nešťastná tečka).

Pak je potřeba použít editoru NE či CSED nebo vestavěného editoru PROLOGU a zjištěné chyby opravit. Po opravě znovu voláme již popsaným způsobem PROLOG a celý cyklus opakujeme, dokud neodstraníme všechny chyby.

Příslušnou DC gramatiku – v našem případě je to `g1.pl` – načteme do databáze PROLOGU příkazem
`[g1]`.
Poznamenáváme, že extenzi (příznak souboru udávající typ souboru) `.pl` není třeba psát, jestliže se jedná o standardní prologovský soubor obsahující výhradně klauzule PROLOGU, případně poznámky uzavřené do speciálních závorek `/* poznámka */`. PROLOG tedy odpoví

```
yes
?-
```
Je tu ještě další možnost, jak zavést DC gramatiku (obecně jakýkoli prologovský soubor) do databáze. Je-li v našem PROLOGU vestavěný predikát `edit(X)` s přiřazeným editorem, stačí na prompt PROLOGU napsat
`?- edit(g1).`,
což způsobí, že daný soubor se uloží do databáze PROLOGU a otevře se pro editování. Ukončíme-li práci se souborem, ať již s jeho uložením nebo bez uložení, bude připraven k další práci stejně jako v předchozím případě.

Nyní jsme připraveni k práci s gramatikou `g1.pl`, tj. můžeme začít s jejím testováním, při kterém si ověřujeme, jak se nám zdařilo popsat příslušné, např. právě české, syntaktické struktury. Naznačili jsme už, že nás zajímají dvě věci:

1. zda je naše gramatika `g1` schopna generovat ty typy vět, které nás zajímají jako předmět našeho popisu.

2. jaké strukturní popisy gramatika `g1` generovaným větám přiřazuje, a zejména, zda jsou v souladu s naším požadavkem deskriptivní (gramatická správnost generovaných vět)) a explikativní adekvátnosti (přiřazované strukturní popisy musí být v souladu s naší lingvistickou intuicí).

3. DC gramatiky v PROLOGU umožňují získat (generovat) na počítači jak 1) tak i 2) a kromě toho PROLOG poskytuje možnost generované strukturní popisy graficky zobrazit v podobě stromových grafů na obrazovce. K tomu slouží program, resp. predikát `draw.pl`, jehož použití bude naznačeno dále.

To, že chceme, aby PROLOG začal generovat podle pravidel `g1` věty a jejich strukturní popisy, sdělíme PROLOGU tak, že mu zadáme cíl pro generování věty v následující podobě:
`s(X,Y,[]).`,
Interpretujeme jej takto: splň predikát `s` tak, že generovanou větu samu uložíš do proměnné `Y` a její strukturní popis, tj. její graf-strom v linearizovaném tvaru tzv. ohodnoceného uzávorkování, do proměnné `X`. `[]` je prázdný seznam (neobsahuje žádné prvky) a pracuje se s ním v průběhu splňování predikátu `s`. Predikát `s` však může být splněn více způsoby, protože je-li generovaná věta víceznačná, odpovídající DC gramatika jí může přiřadit více strukturních popisů – záleží pak na nás, zda chceme vidět všechny či nikoli (viz dále).
Můžeme ale chtít i něco jiného – může nás zajímat nikoli generování vět, nýbrž jejich rozpoznávání (tj. analýza), konkrétně, zda `g1` dovede rozpoznávat určitý typ věty (určité typy vět). Pak musíme PROLOGU sdělit, že má predikát `s` splnit poněkud jiným způsobem: zadáme konkrétní větu, která nás zajímá, třeba (v-1), a napíšeme:
`s(X,[ta,jeho,druhá,žena,vášnivě,miluje,rychlá,auta],[]).`,
což interpretujeme: do `X` ulož linearizovaný strukturní popis věty uvedené v příslušném seznamu následujícím za proměnnou `X`. Prázdný seznam `[]` použij stejným způsobem jako výše.

Poznámka:
Ačkoli jsme zadání konkrétní věty ve výše uvedeném cíli napsali normálně česky, ve skutečnosti je situace s češtinou v PROLOGU poněkud složitější. Slova či v terminologii PROLOGU atomy obsahující česká písmena s diakritickými znaménky nejsou z hlediska PROLOGU standardní. Pokud s nimi v PROLOGU

chceme pracovat, musíme je zvlášť vyznačit tím, že je umístíme mezi dva apostrofy. Teprve pak je PROLOG bude chápat jako standardní atomy. Zadání našeho cíle musí tedy ve skutečnosti vypadat takto:
`s(X,[ta,jeho,'druhá','žena','vášnivě',miluje, 'rychlá',auta],[]).`,
Stejného zápisu musíme pro nestandardní české atomy použít i v gramatice `g1`, jinak PROLOG odmítne cokoli generovat či rozpoznávat.
– Druhou možností je zápis atomů v zadání cíle i uvnitř gramatiky `g1` bez „hacku“ a „carek“: `s(X,[ta,jeho,druha,zena,va`
`rychla,auta],[]).`.
PROLOG tedy po nás vyžaduje, abychom byli ve své volbě konzistentní.

– Třetí možností je doplnit do systému PROLOG příslušné kódování češtiny tak, aby se české atomy mohly psát bez apostrofů. To je ovšem již práce pro šikovného programátora. V současných verzích ARITY PROLOGU není toto řešení implementováno.

Chceme-li v obou uvedených případech, aby PROLOG splnil zadaný cíl jiným způsobem (pokusil se generovat více vět nebo zjistil, zda předloženou větu lze analyzovat více způsoby), napíšeme po prvním splnění cíle „;“ (středník) a po něm stiskneme klávesu `ENTER`. Odpoví-li i potom PROLOG
`no`
`?-`
– znamená to, že cíl již jiným způsobem splnit nelze.

Chceme-li na obrazovce kreslit linearizované strukturní popisy generovaných nebo analyzovaných vět v podobě grafů-stromů, musíme nejprve na prompt
`c:\api>`
napsat buď
`consult(draw).`,
nebo zkráceně
`[draw].`,
což je vhodné udělat hned po spuštění PROLOGU. Kterýmkoli z obou příkazů načteme do PROLOGU program (nebo predikát, což je zde totéž) `draw.pl ( = kresli.pl)`, který je také napsán v PROLOGU, a můžeme jej chápat jako speciální predikát, který dovede v podobě grafu-stromu zobrazit jakýkoli prologovský výraz.[3]

Poté PROLOG vypíše
`?-`
Nyní se můžeme rozhodnout, zda chceme věty generovat a současně s tím i kreslit jejich strukturní popisy. Pokud ano, použijeme příkazu obsahujícího konjunkci dvou predikátů:
`s(X,Y,[]),draw(X).`
Chceme-li jen kreslit strukturní popis věty předložené k analýze, dosáhneme toho pomocí zadání, v němž žádáme jiný způsob splnění predikátu `s` v konjunkci s predikátem `draw`:
`s(X,[ta,jeho,'druhá','žena','vášnivě',miluje,'rychlá', auta],[]),draw(X).`

Uvedeme nyní příklad úplné analýzy a generování jedné české věty včetně zadání a komplet-

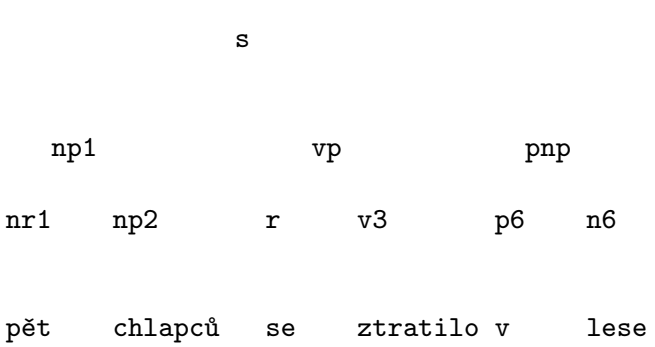
[3] Autorem tohoto programu (predikátu) je dr. J. Gerbrich. Děkuji mu za poskytnutí možnosti pracovat s jeho produktem.

ního výsledku. Použijeme větu z Cvičení 7 uvedeného v oddíle Cvičení, kde čtenář najde také příslušnou gramatiku.

Chceme generovat nebo analyzovat následující větu
(v-23) *Pět chlapců se ztratilo v lese.*
Generování vět, jak jsme již viděli, zadáváme cílem:
`s(X,Y,[]).`
Je-li vše, jak má být, PROLOG ihned generuje větu a její strukturní popis podle gramatiky zadané ve **Cvičení 7** ve tvaru ohodnoceného uzávorkování:
`X = s(np1(nr1('pět'),np2(n2('chlapců'))),vp(r(se),v3(ztratilo),`
`pnp(p6(v), np6(n6(lese)))))`
Budeme-li si přát, aby se linearizovaný strukturní popis vykreslil na obrazovce jako graf-strom, musíme do zadání cíle doplnit predikát `draw(X)`, což učiníme následujícím způsobem (konjunkcí):
`s(X,Y,[]),draw([X]).`

Výsledkem je stromová struktura:

(g-5)



Analýzu zadáváme prakticky stejně, rozdíl je jen v tom, že místo proměnné `Y` se v zadání objeví seznam představující celou vstupní větu:
`s(X,['pět','chlapců',se,ztratilo,v,lese],[]).`
Je-li analýza úspěšná, výsledek se na obrazovce objeví ve stejné podobě jako v případě generování, tedy v té podobě, jak je uvedena výše.

0.6.3 Predikáty `trace` a `spy`

Stane-li se, že při psaní DC gramatiky uděláme chybu, kterou PROLOG při načtení souboru-gramatiky příkazem `[g].` neodhalí, projeví se tato chyba až při generování nebo při analýze zadané věty. Příkladem takové chyby je třeba to, že proměnná začíná malým písmenem (místo velkým, chyba je vyznačena podtržením), např. v pravidle
/*p-1*/ `s(s(np1,VP))` → `np1(Np1),vp(Vp).`

Taková chyba se projeví až při pokusu generovat (nebo analyzovat) zadanou větu, tedy až po zadání cíle `s(X,Y,[]).` tím, že po stisknutí klávesy `ENTER` se na obrazovce prostě vypíše úsečné `no`.

Stejné důsledky má opačná chyba, když třeba predikát omylem začíná velkým písmenem, jak je tomu třeba v následujícím pravidle
/*p-14*/ `np1(np1(Nr1,Np2))` → `Nr1(Nr1), np2(Np2).`
K odhalení takových chyb je potřeba použít speciálních predikátů, které uživateli umožňují sledovat práci programu v PROLOGU a tedy i DC gramatiky krok za krokem. Jsou to predikáty:

- `trace.` – spouští úplné trasování, které uživateli dovoluje sledovat práci daného programu v plném rozsahu. Nevýhodou je, že predikát `trace` poskytuje příliš mnoho informace, takže uživatel může často mít potíže při hledání toho, co skutečně potřebuje.
- `spy(s).` – umístí hlídané body na konkrétní predikát, tj. zde na `s`. Je proto vhodné kombinovat použití predikátu `trace` s tímto predikátem, protože trasování pak probíhá selektivně podle zvoleného predikátu. Chceme-li umístit hlídané body na další predikát, musíme napsat např. `spy(np1)`.
- `notrace.` – ruší trasování.
- `nospy.` – odstraní z predikátu, který byl sledován, hlídané body, tedy zde z predikátu `s`.

Uvedeme nyní příklad použití predikátů `trace` a `spy`.[4] Poté, co jsme již bezchybně načetli příslušnou gramatiku, např. `g3.pl` z Cvičení 5, a to příkazem:
`?- [g3].`,
oznámí PROLOG, že je připraven:
`yes`
`?-`
Nyní chceme analyzovat větu
(v-24) *Chodit za školu se nevyplácí.*
Zadáme tedy cíl
`s(X,[chodit,za,'školu','nevyplácí',se],[]).`
Na to PROLOG suše odpoví
`no`
`?-`
Pokud jsme si nevšimli, že chyba spočívá v tom, že jsme omylem zaměnili pořadí slov – místo *se nevyplácí* jsme napsali *nevyplácí se* – použijeme predikátů pro trasování. Na klávesnici tedy napíšeme:
`?- trace.`
Dostaneme odpověď
`yes`

[4]Na tomto místě je třeba poznamenat, že vše, co je dále uvedeno o predikátech `trace` a `spy` včetně podoby výpisů platí pro ARITY Prolog, kterého jsme užívali při vzniku tohoto textu. V SWI Prologu mají výpisy poněkud jinou podobu a také chování uvedených predikátů je místy mírně odlišné, rozdíly však nejsou podstatné.

```
?-
```
což znamená, že příkaz byl akceptován. Hlídané body umístíme na predikát `s`, neboť správně tušíme, že chyba by mohla být ve slovesné skupině. Napíšeme tedy:
```
spy(s).
```
Také bychom sledovat více predikátů – to bychom třeba vyjádřili pomocí seznamu
```
spy([s,vp]).
```
Odpověd je
```
yes
?-
```
Tím nás PROLOG informuje, že na hlídané body umístil příslušné predikáty.

Nyní znovu zadáme cíl
```
?- s(X,[chodit,za,'školu','nevyplácí',se],[]).
```
a PROLOG začne na obrazovce vypisovat následující informace:

```
?- s(X,[chodit,za,'školu','nevyplácí',se],[])
(api,api) ** (0) CALL: s(_005D,[chodit,za,'školu',
'nevyplácí',se],[]) ? >
(api,api) (1) CALL: npi1(_026D,[chodit,za,'školu', 'nevyplácí',se],_028D) ? >
(api,api) (2) CALL: in(_106,[chodit,za,'školu',
'nevyplácí',se],_111) ? >
(api,api) (2) EXIT: in(in(chodit),[za 'školu','nevyplácí',se], [za,'školu','nevyplácí',se])
(api,api) (3) CALL: pnp(_107,[za,'školu','nevyplácí',se], _104) ? >
(api,api) (4) CALL: p6(_120,[za,'školu','nevyplácí',se], _125) ? >
(api,api) (4) FAIL: p6(_120,[za,'školu','nevyplácí',se], _1225) ? >
(api,api) (5) CALL: p4(_121,[za,'školu','nevyplácí',se], _121) ? >
(api,api) (5) EXIT: p4(p4(za),[za,'školu','nevyplácí',se], ['školu','nevyplácí',se])
(api,api) (6) CALL: n4(_121,['školu','nevyplácí',se],104) ? >
(api,api) (6) EXIT: n4(n4('školu'),['školu','nevyplácí',se], ['nevyplácí',se])
(api,api) (3) EXIT: pnp(pnp(p4(za),n4(n4('školu')),[za,'školu', 'nevyplácí',se],['nevyplácí'
(api,api) (1) EXIT: npi1(npi1(in(chodit),pn(pn(p4(za),n4(n4 ('školu'))),[chodit,za,'školu',ne
(api,api) ** (7) CALL: vp(_100,['nevyplácí',se],[])? >
(api,api) (8) CALL: vr3(_0271,['nevyplácí',se],[])? >
(api,api) (9) CALL: r(_182,['nevyplácí',se],_187)
(api,api) (9) FAIL: r(_182,['nevyplácí',se],_187)
(api,api) (8) FAIL: vr3(_40,['nevyplácí',se],[]) ? >
(api,api) ** (7) FAIL: vp(_401,['nevyplácí',se],[]) ?
(api,api) (1) REDO: npi1(npi1(in(chodit),pnp(p4(za),n4('školu'))), [chodit,za,'školu','nevypl
? >
(api,api) (3) REDO: pnp(pnp(p4(za),n4('školu')), [za,'školu','nevyplácí',se],['nevyplácí',se]
? >
(api,api) (6) REDO: n4(n4('školu'),['školu','nevyplácí',se],
('nevyplácí',se)) ? >
(api,api) (6) FAIL: n4(_0711,['školu','nevyplácí',se], _028D) >
(api,api) (5) REDO: p4(p4(za),[za,'školu','nevyplácí',se],['školu',
```

```
'nevyplácí',se]) ? >
(api,api) (5) FAIL: p4(_0701,[za,'školu','nevyplácí',se], _072D) ? >
(api,api) (3) FAIL: pnp(_0370,[za,'školu','nevyplácí',se], _022D) ? >
(api,api) (2) REDO: in(in(chodit),[chodit,za,'školu','nevyplácí',
se],[za,'školu','nevyplácí',se]) ? >
(api,api) (2) FAIL:in(_03C0,[,chodit,za,'školu','nevyplácí',
se],_042D) ? >
(api,api) (1) FAIL: npi1(_03C9,[,chodit,za,'školu','nevyplácí',
se],_028D) ? >
(api,api) (0) FAIL: s(_005D,[,chodit,za,'školu','nevyplácí',se],[])? >
no
?-
```

Co můžeme z hlášení PROLOGU vyčíst?[5] Vidíme, že PROLOG se snaží splnit cíl (predikát) `s` (tj. analyzovat zadanou větu). Nejprve však musí splnit cíl (predikát) `npi1`, tj. analyzovat jmennou skupinu s infinitivem, který se rozpadá na podcíle `in` – nalezení infinitivu a `pnp` – analýzu předložkové skupiny. Podcíl `pnp` obsahuje ještě další podcíle `p4` – analýzu předložky a `n4` – analýzu substantiva v akuzativu. Podíváme-li se na jednotlivé řádky s těmito cíli, uvidíme, že je u nich napsáno `EXIT (= výstup)`, což nám říká, že uvedené cíle byly úspěšně splněny. Zbývá ještě splnit cíl `vp`, tj. analyzovat slovesnou skupinu ve větě, který se rozpadá na podcíle `r` – nalezení zvratného *se* a `v3r` – nalezení určitého slovesného tvaru, což je také dobře vidět z pravidel gramatiky `g5` (viz Cvičení 5). U cíle `r` a posléze i u cíle `vp` a následně též dalších se však objevuje hlášení `(9) FAIL :`, které znamená, že tyto cíle `neuspěly`. Cíl `vp` pochopitelně musel selhat, protože ve vstupní větě jsme omylem zaměnili pořadí slov, a v důsledku toho i složek `v3r` a `r`. V `g5` však nacházíme jen pravidlo
/*p-15*/ `vp(vp(R,V3r)) → r(R),v3r(V3r).`,
které nekompromisně vyžaduje, aby ve vstupní větě bylo pořadí slov *se nevyplácí*.

Popisovanou chybu lze odstranit buď tak, že analyzovanou větu (v-24) uvedeme v zadání cíle se správným slovosledem, nebo do gramatiky `g5` doplníme pravidlo
/*p-15a*/ `vp(vp(V3r,R)) → v3r(V3r),r(R).`

Gramatika `g5` bude v takovém případě schopna analyzovat (v-24) s oběma zmíněnými slovosledy. Práci v PROLOGU ukončíme příkazem
`halt.`
– nebo ve verzi 5.1 volbou příslušné položky v menu. Na obrazovce se poté objeví prompt operačního systému: `C:\API>` Je-li k našemu počítači připojena tiskárna, existuje možnost výsledky generování a analýzy vytisknout.

Poznámka: Do PROLOGU je však nutno doplnit příkaz (predikát) `anal`. Pokud s ním chceme pracovat, musíme na konec své gramatiky dopsat jeho definici:

```
anal :- anal(con,con).
anal(Vystup) :- anal(con,Vystup).
anal(Vstup,Vystup) :-
        see(Vstup),
```

[5] Hlášení, které se při konkrétním trasování objeví na vaší obrazovce, je ve skutečnosti o něco delší, neboť obsahuje všechna navracení, tj. všechny pokusy splnit zadaný cíl všemi možnými způsoby. Zde jsme uvedli jen hlavní pokusy – `REDO`, tj. `předělej`.

```
        ( ( Vystup \= con , create(Handle,Vystup), assert(filex(Handle)) )
          ; true ),
        cls, at(15,1),
        repeat, analyza,
        see(Vstup), seen, tell(Vystup), told, !, retract(filex(_)).
anal(_,_).
```

`anal` umožňuje uložit výsledky generování a analýzy do samostatného souboru, který lze posléze vytisknout v rámci operačního systému MS DOS příkazem:

`C:\API> print jméno_souboru`.

Je ovšem pochopitelné, že před použitím tohoto příkazu musí být tiskárna zapnuta a připravena k tisku, tedy ve stavu `ONLINE`.

0.7 Struktura české věty II

V předchozích oddílech jsme se seznámili se základními principy formálního popisu syntaktické struktury české věty a ukázali jsme, jak je uplatnit v rámci nekontextových gramatik. Umíme také převádět nekontextové gramatiky na DC gramatiky v PROLOGU a testovat je na počítači. Příklady, s nimiž jsme dosud pracovali, byly však velmi jednoduché a představovaly především ilustraci základních principů formálního popisu českých syntaktických struktur. Navíc některé z nich, jak dále ukážeme, nejsou zcela korektní. Gramatika `g1.pl` generuje řadu gramaticky nesprávných vět – pokusíme se analyzovat, proč tomu tak je a jaká by mohla být náprava. Vyjdeme nyní z toho, co již známe, a pokusíme se pokročit dále: s použitím DC gramatik uvedeme příklady adekvátnějšího popisu českých syntaktických struktur a nastíníme řešení problémů, které v této souvislosti vznikají.

0.7.1 Gramatická shoda a pády

Není těžké vidět, že výše uvedená `g1.pl` dovoluje generovat gramaticky nesprávné české věty. Způsobují to např. pravidla (`p-2`), (`p-2a`), (`p-2b`) a samozřejmě první pravidlo (`p-1`). Snadno se o tom přesvědčíme, když pravidlo (`p-9`) doplníme o další řádky, např.
`n1(n1(syn))` → `[syn].`
nebo
`n1(n1(děda))` → `[děda].`
Čtenář se snadno může přesvědčit, že po této úpravě bude gramatika `g1.pl` generovat věty jako
Ta *jeho syn bláznivě miluje silná auta.*
A přidáme-li do pravidla (`p-10`) ještě řádek
`v3(v3(milovali))` → `[milovali].`,
bude `g1.pl` generovat (nebo rozpoznávat) věty jako
Ta jeho chytrá žena vášnivě milovali *rychlá auta.*

Jak vidíme, `g1.pl` není schopna postihnout kontextové závislosti uvnitř větné struktury a dovoluje generovat věty, v nichž je porušena gramatická shoda jednak uvnitř jmenné skupiny `np1` a jednak mezi `np1` a `vp`. Jinými slovy, pravidla v dosud uvažované DC gramatice nepostačují k adekvátnímu popisu gramatické shody (v čísle a rodě a v čísle a osobě), postihují však shodu v pádě – toho je dosaženo rozlišením jmenných skupin `np1, n1` a `np4, n4`. Podobného postupu bychom mohli použít i pro gramatické významy čísla, rodu a osoby, ale cítíme jistě, že toto řešení by bylo notačně poněkud komplikované, a ne vždy zcela systematické. Pozorný čtenář si již zajisté uvědomil, že uvedené výhrady se vztahují nejen na DC gramatiku `g1.pl`, ale zejména na výchozí nekontextovou gramatiku **g1**. Jako cvičení doporučujeme čtenáři, aby se pokusil upravit příslušná pravidla **g1** tak, že **g1** bude postihovat gramatickou shodu.

Na rozdíl od nekontextových gramatik poskytuje naštěstí formální aparát DC gramatik teoreticky úplné a korektní řešení naznačené situace a problémů, které z ní vyplývají.

Víme již, že v DC gramatikách odpovídají neterminálním symbolům predikáty se svými argumenty. Hledané řešení našeho problému tedy spočívá v tom, že gramatické významy pádu, čísla, rodu a osoby vyjádříme pomocí argumentů, které přidáme k těm jednotlivým predikátům, u nichž požadujeme splňování pravidel gramatické shody.

Budeme tak vlastně vytvářet novou verzi gramatiky `g1.pl`, kterou pojmenujeme `g2.pl`. Odpovídající si pravidla jsou v obou gramatikách označena stejnými čísly. Víme už, že inkriminovanými predikáty jsou `np` a `vp`, které se vyskytují v řadě pravidel. Pro jednotlivé gramatické významy si zavedeme následující označení:
pád – `K`, číslo – `Nu`, rod – `G`, osoba – `P`.

Tyto symboly přidáme k predikátům `np` a `vp` jako argumenty – proměnné, které mohou nabývat různých hodnot – u pádu `K` jde o `1.,...,7.` pád, u čísla `Nu` o `sg.` a `pl.`, u rodu `G` o `mz` (mužský životný), `mn` (mužský neživotný), `f` (ženský), `n` (střední) a u osoby `P` o `1., 2., 3.` (osobu).

Původní první pravidlo z `g1.pl` bude nyní vypadat takto:
`/* p-1 */ s(s(Np,Vp)) → np(Np,P,1,Nu,G), vp(Vp,P,K,Nu,G).`,
což můžeme interpretovat tak, že predikát `s` je splněn, jestliže jsou splněny predikáty `np` a `vp` tak, že jejich jednotlivé argumenty (proměnné) nabudou stejných hodnot. Jinak (lingvisticky) řečeno, pravidlo `p-1` nyní postihuje všechny případy gramatické shody, které se mohou vyskytnout mezi podmětovou a přísudkovou částí věty v `g2.pl`. Navíc, jak si pozorný čtenář(ka) už jistě všiml(a), číselný argument, tj. konstanta `1`, zde reprezentuje konkrétní hodnotu proměnné `K`, jíž je konkrétní pád, a to `nominativ`. Tímto způsobem lze jednoduše a elegantně zachytit skutečnost, že definujícím příznakem podmětové části věty je její nominativní forma.

Podobně upravíme i další pravidla, v nichž se vyskytují predikáty `np` a `vp`. Stejně jako jsme u pravidla `/* p-1 */` v predikátu `np` požadovali nominativ (`1`), můžeme v pravidle `/* p-3b */` u téhož predikátu vyznačit akuzativ (`4`), a tak dosáhnout toho, že se v našich větách bude správně generovat i analyzovat u příslušných sloves akuzativní rekce. Podobně je, jak vidno, formulováno i pravidlo `/* p-3c */`, i když v něm již nejde o zachycení rekce, ale příslušného adverbiálního pádu.

Naznačený postup má obecnou platnost a je tedy vhodný pro popis kteréhokoli přirozeného jazyka, v němž se setkáváme s kontextovými vztahy v rámci věty. Lze tedy závěrem říci, že DC gramatiky představují `adekvátnější` formální nástroj pro popis syntaktických (ale i morfologických a sémantických) struktur než nekontextové gramatiky. Navíc – díky `prologu` – jsou i počítačovými programy, které můžeme podle potřeby testovat.

Aby si čtenář mohl učinit představu o vztazích jednotlivých pravidel, uvádíme celou gramatiku `g2.pl`, která může sloužit jako vzor pro budování podobných (a složitějších) gramatik:

```
/*p-2*/ np(np(N),_,K,Nu,G) → n(N,K,Nu,G).
/*p-2a*/ np(np(Pnd,N),_,K,Nu,G) → pnd(Pnd,K,Nu,G),n(N,K,Nu,G).
/*p-2b*/ np(np(A,N),_,K,Nu,G) → a(A,K,Nu,G),n(N,K,Nu,G).
/*p-2c*/ np(np(Pos,N),_,K,Nu,G) → pos(Pos,K,Nu,G),n(N,K,Nu,G).
/*p-2d*/ np(np(Numo,N),_,K,Nu,G) → numo(Numo,K,Nu,G),n(N,K,Nu,G).
/*p-3*/ vp(vp(V),P,K,Nu,G) → v(V,P,K,Nu,G).
/*p-3b*/ vp(vp(V,Np),P,K,Nu,G) → v(V,P,K,Nu,G),np(Np,_,4,_,v_).
/*p-3a*/ vp(vp(Adp,V),P,K,Nu,G) → adp(Adp),v(V,P,K,Nu,G).
/*p-3c*/ vp(vp(V,Pp),P,K,Nu,G) → v(V,P,K,Nu,G),pp(Pp,6).
/*p-4*/ adp(adp(Adm)) → adm(Adm).
/*p-5*/ pp(pp(Pre,Np),K) → pre(Pre,K),np(Np,_,K,_,_).

/* slovnik */
/*p-6*/ pnd(pnd(ta),1,sg,f) → [ta].
        pnd(pnd(ta),4,sg,mn) → [ten].
        pnd(pnd(ten),1,sg,mz) → [ten].
/*p-7*/ pos(pos(jeho),_,_,_) → [jeho].
        pos(pos(moje),1,sg,f) → [moje].
```

```
         pos(pos(muj),1,sg,mz) → [muj].
         pos(pos(muj),1,sg,mn) → [muj].
         pos(pos(muj),4,sg,mn) → [muj].
/*p-8*/ numo(numo(druha),1,sg,f) → [druha].
         numo(numo(druhy),1,sg,mz) → [druhy].
/*p-9*/ n(n(zena),1,sg,f) → [zena].
          n(n(syn),1,sg,mz) → [syn].
          n(n(auta),1,pl,n) → [auta].
          n(n(auta),4,pl,n) → [auta].
          n(n(aute),6,sg,n) → [aute].
          n(n(kurata),1,pl,n) → [kurata].
          n(n(kurata),4,pl,n) → [kurata].
          n(n(mladi),6,pl,n) → [mladi].
/*p-10*/ v(v(miluje),3,1,sg,) → [miluje].
          v(v(milovala),3,_,sg,f) → [milovala].
          v(v(miloval),3,_,sg,mz) → [miloval].
          v(v(nenavidi),3,1,sg,_) → [nenavidi].
/*p-11*/ a(a(krasna),1,sg,f) → [krasna].
         a(a(krasna),1,pl,n) → [krasna].
          a(a(krasna),4,pl,n) → [krasna].
          a(a(chytra),1,sg,f) → [chytra].
          a(a(rychla),1,sg,f) → [rychla].
          a(a(rychla),1,pl,n) → [rychla].
          a(a(rychla),4,pl,n) → [rychla].
          a(a(silna),4,pl,n) → [silna].
/*p-12*/ adm(adm(vasnive)) → [vasnive].
          adm(adm(silene)) → [silene].
/*p-13*/ pre(pre(v),6) → [v].

/* pro doplnění uvádíme možnou rekurzívní variantu pravidla p-2 */
/*p-2e*/ np(np(A,Np),_,K,Nu,G) → a(A,K,Nu,G),np(Np,K,Nu,G).
```

Poznámka:
Symbol _ označuje tzv. anonymní proměnnou, u níž nám nezáleží na tom, jaké hodnoty nabude při splňování příslušného predikátu.

0.7.2 Slovosled

Speciálním problémem v češtině je slovosled. Jak známo, jedna česká syntaktická struktura může mít řadu variant daných obměnami slovosledu, přičemž někdy jsou přípustné téměř všechny možné permutace. Vytváření slovosledných variant lze popsat pomocí poměrně složitého systému pravidel (viz např Karlík, Svoboda, 1985), který má trojstupňový charakter – obsahuje pravidla rytmická (pro kladení příklonek), gramatickosémantická (fungující uvnitř větných složek) a pravidla funkční perspektivy větné (postihující pořadí složek ve větném kon-

textu).

Je tedy přirozené položit si otázku, jak se s variabilitou českého slovosledu vyrovnáme v aparátu DC gramatik. Odpověď je poměrně snadná, ale neuspokojivá: se značnými obtížemi.

Přesněji řečeno, DC pravidly lze snadno popsat všechny možné variace jedné syntaktické struktury obsahující třeba čtyři větné složky `np, vp, adp, pp`, musíme to ovšem udělat tak, že formulujeme tolik DC pravidel, kolik daná struktura připouští slovosledných variant. Nepříjemné na tom je pouze to, že takových variant může někdy být opravdu hodně a vzniklá DC gramatika je pak značně rozsáhlá, a proto i nepřehledná a špatně čitelná.

Je vcelku zřejmé, že tu jde především o problém ani ne tak lingvistický, jako spíše programátorský – DC gramatiky je potřeba modifikovat tak, aby připouštěly např. pravidla jako
`/* p-sl */ s(s(Np,Vp)` → `{np(Np),vp(Vp)}.`,
v nichž svorky vyznačují nikoli řetězy, nýbrž množiny. Taková modifikace DC gramatik existuje a nese název mezerové (gapping) gramatiky (Dahl, Abramson, 1984), jejich implementace nám však v současnosti bohužel není dostupná, takže ji v rámci tohoto textu nemůžeme nabídnout.

Pro potřeby experimentů s větší českou DC gramatikou (Halasová, Pala, Franc, 1987) byly vyvinuty modifikace DC pravidel umožňující v jednom pravidle zachytit řadu slovosledných variant, použité řešení však nebylo dostatečně obecné, proto je zde neuvádíme. Překonání těchto obtíží je závislé na užší spolupráci s programátory.

0.8 Sémantická struktura české věty

Obrátíme nyní pozornost k popisu sémantické stavby české věty v rámci DC gramatik. Vyjdeme přitom z poznatků, které jsme již získali při popisu syntaktické roviny a českých syntaktických struktur.

Dosavadní pravidla v příkladových gramatikách `g1, g2` obsahovala jako nejvyšší větné složky `np, vp, adp, pp, s`, které jsou vymezeny na základě svých formálních vlastností. Vypovídají tedy velmi málo (prakticky nic) o tom, jaké významové charakteristiky by se s nimi mohly spojovat. Jinými slovy, stojíme před otázkou, zda neexistuje možnost definovat takové větné složky, pomocí nichž bychom mohli popisovat významovou stavbu (českých, anglických, německých, ...) vět a které by byly nějak nadřazeny již zmíněným složkám „syntaktickým". Odpověď na tuto otázku je kladná a vede k zavedení dvou druhů přepisovacích pravidel: jedněch, která prostřednictvím sémantických složek popisují významovou stavbu věty na nejvyšší úrovni, a druhých, která na ně navazují a slouží k charakterizování syntaktické stavby věty. Dodejme ještě, že nám nic nebrání v tom, abychom šli ještě o rovinu níž a způsobem naznačeným výše začlenili do našeho popisu i pravidla morfologická. V rámci DC gramatik lze takto vytvořit integrovaný popis české věty zahrnující rovinu morfologickou, syntaktickou i sémantickou. Čtenář se může sám pokusit o formulování takové nevelké integrované DC gramatiky, v zásadě jde více o problém technický (implementační a programátorský) než lingvistický.

0.8.1 Sémantické pády v DC gramatice

Teoretické řešení, které v tomto kontextu hledáme, poskytuje teorie sémantických pádů (aktantů, participantů, sémantických rolí, Fillmore, 1968). Je založena na myšlence, že každé sloveso se v závislosti na svém významu pojí s určitými sémantickými pády (rolemi, participanty) chápanými jako nominální (substantivní) elementy relevantní pro sémantickou klasifikaci sloves. Kombinaci sémantických pádů u daného slovesa pak nazýváme jeho pádovým rámcem. Podle těchto pádových rámců se pak slovesa seskupují do přirozených významových tříd a vytvářejí sémantickou klasifikaci sloves. Každý pádový rámec popisuje vlastně významovou stavbu věty s daným slovesem a může mít podobu přesně definovaného formálního pravidla. Lze také říci, že pádové rámce představují prostředek pro charakterizování toho, čemu se v teorii generativních gramatik říká hloubková struktura věty (na rozdíl od její struktury povrchové, která je dána prostřednictvím složek odkazujících k vlastní syntaktické struktuře věty).

Obvykle se pracuje s poměrně malým počtem sémantických pádů, např. u Fillmora jich najdeme kolem 10, ovšem někteří (Sgall a kol., 1986) jich zavádějí až 30. Popis s větším počtem participantů je nepochybně adekvátnější, i když ne tak obecný a elegantní, nicméně se zdá, že pro konkrétní aplikace se vždy volí jistý druh kompromisu (kolem 20 pádů).

Zde budeme rozlišovat

1. obligatorní (vnitřní) participanty, k nimž řadíme agens (`Ag`), objektiv (`Ob`), adresát (`Adr`), patiens (`Pat`), případně i některé další (zdroj informace, příjemce informace),

2. fakultativní participanty jako beneficient (`Ben`), instrument (`Ins`), sociativ (`Soc`), původ (`Orig`), kauzativ (`Caus`),

3. volné modifikátory, k nimž zcela jistě patří způsob (M), lokativ (L), temporativ (T), komparativ (Komp), ale ev. i další.

Jako příklad uvedeme nyní DC gramatiku g3.pl, která se dosti liší od g2.pl v tom, že její jednotlivá pravidla jsou ve skutečnosti DC modifikacemi pádových rámců pro slovesa patřící do významové třídy v1, tj. pro některá slovesa označující dávání *dávat, prodávat, předávat*. Díky tomuto postupu se ovšem obejdeme bez složky vp a věta s se člení vždy na sloveso z třídy v1 a příslušnou variantu pádového rámce obsahující sémantické pády Ag, Ob, Adr, T, L, M.

Ke gramatice g3.pl jsme neuvedli slovníková pravidla. Učinili jsme tak záměrně a čtenář si, inspirován g2.pl, může slovník formou cvičení vytvořit sám.

g3.pl

```
s(s(V1)) → v1(V1,P,K,Nu,G).

s(s(Ag,V1)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G).
s(s(V1,Ag)) → v1(V1,P,K,Nu,G),ag(Ag,P,K,Nu,G).
s(s(V1,Ob)) → v1(V1,P,K,Nu,G),ob(Ob).
s(s(Ob,V1)) → ob(Ob),v1(V1,P,K,Nu,G).

s(s(V1,T)) → v1(V1,P,K,Nu,G),t(T).
s(s(T,V1)) → t(T),v1(V1,P,K,Nu,G).
s(s(V1,L)) → v1(V1,P,K,Nu,G),l(L).
s(s(L,V1)) → l(L),v1(V1,P,K,Nu,G).
s(s(V1,M)) → v1(V1,P,K,Nu,G),m(M).
s(s(M,V1)) → m(M),v1(V1,P,K,Nu,G).

s(s(T,Ag,V1)) → t(T),ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G).
s(s(Ag,V1,L)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),l(L).
s(s(Ag,V1,T)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),t(T).
s(s(Ag,V1,M)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),m(M).
s(s(Ag,M,V1)) → ag(Ag,P,K,Nu,G),m(M),v1(V1,P,K,Nu,G).

s(s(Ag,V1,Ob)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),ob(Ob).
s(s(Ob,V1,Ag)) → ob(Ob),v1(V1,P,K,Nu,G),ag(Ag,P,K,Nu,G).
s(s(Ob,Ag,V1))8 → ob(Ob),ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G).

s(s(Ag,V1,Ob,Adr)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),ob(Ob),adr(Adr).
s(s(Ag,V1,Adr,Ob)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),adr(Adr),ob(Ob).
s(s(Adr,V1,Ob,Ag)) → adr(Adr),v1(V1,P,K,Nu,G),ob(Ob),ag(Ag,P,K,Nu,G).

s(s(Ag,V1,Ob,L)) → ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),ob(Ob),l(L).
s(s(L,Ag,V1,Ob)) → l(L),ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),ob(Ob).
s(s(Ag,V1,Adr,Ob,L)) →            ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),adr(Adr),ob(Ob),l(L).
```

```
s(s(Ag,T,V1,Ob,Adr)) →
          ag(Ag,P,K,Nu,G),t(T),v1(V1,P,K,Nu,G),ob(Ob),adr(Adr).
s(s(T,Ag,V1,Ob,Adr)) →
          t(T),ag(Ag,P,K,Nu,G),v1(V1,P,K,Nu,G),ob(Ob),adr(Adr).

ag(ag(Np),3,K,Nu,G) → np(Np,1,Nu,G,TL).
ob(op(Np)) → np(Np,4,Nu,G,TL).
np(np(Pos,N),_,K,Nu,G) → pos(Pos,K,Nu,G),n(N,K,Nu,G).
```

0.8.2 Struktura slovníku

Pohled na slovníková pravidla v DC gramatice `g2.pl` ukazuje, že jednotlivá pravidla obsahují údaje o slovním druhu daného slova (slovního tvaru) a také gramatické významy, které se s daným tvarem pojí – pád, číslo, rod, osobu. Přidáním dalších argumentů k predikátům reprezentujícím jednotlivé slovní druhy bychom do jednotlivých pravidel mohli v případě potřeby doplnit ještě další gramatické významy, např. vid, slovesný způsob, čas aj., čímž bychom získali jistý celek obsahující prakticky všechny syntakticky relevantní údaje s výjimkou sémantických pádů (participantů). Údaje o participantech jsou však již obsaženy v pádových rámcích zakódovaných v jednotlivých pravidlech DC gramatiky `g3`.

Víme však, že hesla v normálních jednojazyčných slovnících, jakým je např. Slovník spisovného jazyka českého, 1989, mají podstatně složitější strukturu a obsahují heslové slovo v základním tvaru včetně definice významu slova. Náš způsob organizace slovníku vycházející z jednotlivých slovních tvarů je, jak patrno, značně redundantní a řada údajů se v našem slovníku zbytečně opakuje.

Je tedy jasné, že v jazyce s bohatou morfologií, jakým je právě čeština, musíme v případě většího slovníku počítat s morfologickou analýzou a do slovníku ukládat jen kmeny či kořeny slov. Tohoto řešení jsme ostatně použili jednak u integrovaného analyzátoru KLARA (Halasová, Franc, Pala, 1987) a také při budování velkého strojového slovníku českých kmenů, který čítá kolem 170 000 kmenů (Osolsobě, Franc, Pala, 1990) a je jádrem českého automatického korektoru implementovaného např. v textovém procesoru T602. Pro jednoduché experimenty, na nichž si chceme ověřit fragmenty popisu českých syntaktických a sémantických struktur však postup použitý v `g2.pl` plně postačuje.

Úplný strojový slovník, který by obsahoval stejné množství údajů jako např. Slovník spisovného jazyka českého, 1989, by musel mít podstatně složitější strukturu hesla než slovníky, které jsme zatím popisovali. Heslo v takovém slovníku se bude členit na část morfologickou – obsahující veškeré informace pro automatickou morfologickou analýzu heslového slova, syntaktickou – zahrnující údaje o slovním druhu, gramatických významech a kombinatorice slova (u sloves např. údaje o sémantických pádech) a sémantickou, v níž se musí nacházet informace o významu lexému a významových vztazích k ostatním lexémům ve slovníku včetně víceznačnosti.

V současné počítačové lexikografii (viz např. Boguraev, Briscoe, 1989) se postupuje tak, že ze současných slovníků se vytvářejí tzv. počítačově čitelné slovníky, které pak slouží jako zdroje pro sestavování rozličných typů slovníků včetně specializovaných slovníků pro počítačové zpra-

cování přirozeného jazyka. Pokud jde o češtinu, bude v brzké době k dispozici střední Slovník spisovné češtiny, 1978 ve strojové čitelné podobě, což umožní vytvářet adekvátnější počítačové slovníky zejména s orientací na popis významových vztahů mezi heslovými slovy.

0.9 Cvičení

/*Cvičení 1:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Tisíce lidí provolávalo slávu vítězům.*“*/

```
/*1*/   s(s(Np1,Vp))              →        np1(Np1),vp(Vp).
/*2*/   vp(vp(V3,Np4))            →        v3(V3),np4(Np4).
/2*3*/ np4(np4(N4))               →        n4(N4).
```

/*Řešení 1:
Vytvoříme klauzuli pro podmět vyjádřený číslovkou a substantivem v genitivu. Vytvoříme klauzuli pro rozšíření `vp` tak, aby obsahovala `np4` i `np3`. Doplníme slovník o substantiva v akuzativu, genitivu a dativu.*/

```
/*4*/   np1(np1(Nr,Np2))          →        nr(Nr),np2(Np2).
        np2(np2(N2))              →        n2(N2).
/*5*/   vp(vp(V3,Np4,Np3))        →        v3(V3),np4(Np4),np3(Np3).
        np3(np3(N3))              →        n3(N3).
        nr(nr('pět'))             →        ['pět'].
        nr(nr('tisíce'))          →        ['tisíce'].
        n2(n2('lidí'))            →        ['lidí'].
        n2(n2('chlapců'))         →        ['chlapců'].
        n3(n3('vítězům'))         →        ['vítězům'].
        n3(n3('vítězi'))          →        ['vítězi'].
        n4(n4('slávu'))           →        ['slávu'].
        v3(v3('provolávalo'))     →        ['provolávalo'].
```

/*Cvičení 2:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Být, či nebýt, to je ta otázka.*“*/

```
/*1*/   s(s(Np1,Vp))              →        np1(Np1),vp(Vp).
/*2*/   np1(np1(N1))              →        n1(N1).
```

/*Řešení 2:
Vytvoříme klauzuli pro věty, které mají podmět vyjádřen infinitivem, resp. dvěma infinitivy spojenými souřadně spojkou vylučovací, a přísudek jmenný se sponou. Doplníme slovník.*/

```
/*3*/   np1(np1(In1,K,In2,Pnd1n))     →        in1(In1),k(K),in2(In2),pnd1n(Pnd1n).
/*4*/   vp(vp(Vb3,Nps))               →        vb3(Vb3),nps(Nps).
        nps(nps(N1))                  →        n1(N1).
        nps(nps(Pnd1,N1))             →        pnd1(Pnd1),n1(N1).
        n1(n1('otázka'))              →        ['otázka'].
        vb3(vb3(je))                  →        [je].
        pnd1(pnd1(ta))                →        [ta].
```

```
pnd1n(pnd1n(to))          →      [to].
in1(in1('být'))           →      ['být'].
in2(in2('nebýt'))         →      ['nebýt'].
k(k('či'))                →      ['či'].
```

/*Cvičení 3:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Kdo šetří, má za tři.*“*/

```
/*1*/  s(s(Np1,Vp))          →      np1(Np1),vp(Vp).
```

/*Řešení 3:
Vytvoříme klauzuli pro věty, které mají podmět vyjádřen vedlejší větou vztažnou. Doplníme klauzuli pro přísudek. Doplníme slovník.*/

```
/*2*/  s(s(Skd,Vp))          →      skd(Skd),vp(Vp).
/*3*/  skd(skd(Pronr,V3))    →      pronr(Pronr),v3(V3).
/*4*/  vp(vp(V3,PNr4))       →      v3(V3),pnr4(PNr4).
       pnr4(pnr4(P4,Nr4))    →      p4(P4),nr4(Nr4).
       p4(p4(za))            →      [za].
       nr4(nr4('tři'))       →      ['tři'].
       pronr(pronr(kdo))     →      [kdo].
       v3(v3('má'))          →      ['má'].
       v3(v3('šetří'))       →      ['šetří'].
```

/*Cvičení 4:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „Otec s matkou šli do kina.“*/

```
/*1*/  s(s(Np1,Vp))          →      np1(Np1),vp(Vp).
```

/*Řešení 4: Vytvoříme klauzuli pro podmět. Vytvoříme klauzuli pro přísudek a pro příslovečné určení místa. Doplníme slovník.*/

```
/*2*/  np1(np1(N1,PNp7))     →      n1(N1),pnp7(PNp7).
       pnp7(pnp7(P7,Np7))    →      p7(P7),np7(Np7).
       np7(np7(N7))          →      n7(N7).
/*3*/  vp(vp(V3,PNp4))       →      v3(V3),pnp4(PNp4).
       pnp4(pnp4(P4,Np4))    →      p4(P4),np4(Np4).
       np4(np4(N4))          →      n4(N4).
       n1(n1(otec))          →      [otec].
       n4(n4(kina))          →      [kina].
       n7(n7(matkou))        →      [matkou].
       p4(p4(do))            →      [do].
       p7(p7(s))             →      [s].
       v3(v3('šli'))         →      ['šli'].
```

/*Cvičení 5:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Chodit za školu se nevyplácí.*“*/

```
/*1*/   s(s(Np1,Vp))              →        np1(Np1),vp(Vp).
/*2*/   np1(np1(N1))              →        n1(N1).
```

/*Řešení 5:
Vytvoříme klauzuli pro věty, které mají podmět je vyjádřen infinitivem. Doplníme pravidlo pro přísudek. Doplníme do slovníku infinitivy, předložkové `np`, adjektiva, slovesa.

```
/*3*/   s(s(Npi1,Vp))             →        npi1(Npi1),vp(Vp).
/*4*/   npi1(npi1(In))            →        in(In).
/*5*/   npi1(npi1(In,Pnp))        →        in(In),pnp(Pnp).
/*6*/   pnp(pnp(P,Np))            →        p(P),np(Np).
        pnp(pnp(P6,N6))           →        p6(P6),n6(N6).
        pnp(pnp(P4,N4))           →        p4(P4),n4(N4).
/*7*/   vp(vp(Vr3))               →        vr3(Vr3).
        vr3(vr3(R,V3r))           →        r(R),v3r(V3r).
        in(in(pracovat))          →        [pracovat].
        in(in(chodit))            →        [chodit].
        p6(p6(na))                →        [na].
        p4(p4(za))                →        [za].
        n6(n6(poli))              →        [poli].
        n4(n4('školu'))           →        ['školu'].
        v3r(v3r('nevyplácí'))     →        ['nevyplácí'].
        r(r(se))                  →        [se].
```

/*Cvičení 6:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Petr chodí do školy a Pavel běhá za děvčaty.*“*/

```
/*1*/   s(s(Np1,Vp))              →        np1(Np1),vp(Vp).
        np1(np1(N1))              →        n1(N1).
        vp(vp(V3,PNp2))           →        v3(V3),pnp2(PNp2).
        pnp2(pnp2(P2,Np2))        →        p2(P2),np2(Np2).
        np2(np2(N2))              →        n2(N2).
```

/*Řešení 6:
Vytvoříme klauzuli pro souvětí slučovací a doplníme slovník.*/

```
/*2*/   ss(ss(S,KS1))             →        s(S),ksa(KSa).
        ksa(ksa(K,Sa))            →        k(K),sa(Sa).
        sa(sa(Np1a,Vpa))          →        np1a(Np1a),vpa(Vpa).
        np1a(np1a(Na1))           →        n1(Na1).
        vpa(vpa(Va3,PNpa7))       →        v3(Va3),pnp7(PNpa7).
```

```
pnp7(pnp7(Pa7,Npa7))     →      p7(Pa7),np7(Npa7).
np7(np7(Na7))            →      n7(Na7).
n1(n1(otec))             →      [otec].
n1(n1('Pavel'))          →      ['Pavel'].
n1(n1('Petr'))           →      ['Petr'].
n2(n2(kina))             →      [kina].
n2(n2('školy'))          →      ['školy'].
n7(n7(matkou))           →      [matkou].
n7(n7('děvčaty'))        →      ['děvčaty'].
p2(p2(do))               →      [do].
p7(p7(s))                →      [s].
p7(p7(za))               →      [za].
v3(v3('šel'))            →      ['šel'].
v3(v3('chodí'))          →      ['chodí'].
v3(v3('běhá'))           →      ['běhá'].
k(k(a))                  →      [a].
```

/*Cvičení 7:
Doplň následující gramatiku tak, aby bylo možné generovat věty typu: „*Pět chlapců se ztratilo v lese.*“*/

```
/*1*/  s(s(Np1,Vp))          →      np1(Np1),vp(Vp).
/*2*/  np1(np1(N1))          →      n1(N1).
/*3*/  vp(vp(V3,Np4))        →      v3(V3),np4(Np4).
/*4*/  np4(np4(N4))          →      n4(N4).
```

/*Řešení 7:
Podmět je vyjádřen číslovkou v nominativu a podstatným jménem v genitivu. Vytvoříme prologovské pravidlo pro tento typ podmětu. Vytvoříme pravidlo pro `vp` vyjádřenou zvratným slovesem v určitém tvaru a příslovečným určením. Doplníme slovník.*/

```
/*5*/  np1(np1(Nr1,Np2))     →      nr1(Nr1),np2(Np2).
       np2(np2(N2))          →      n2(N2).
/*6*/  vp(vp(R,V3,PNp))      →      r(R),v3(V3),pnp(PNp).
       pnp(pnp(P6,Np6))      →      p6(P6),np6(Np6).
       np6(np6(N6))          →      n6(N6).
       nr1(nr1('pět'))       →      ['pět'].
       n2(n2('chlapců'))     →      ['chlapců'].
       n6(n6(lese))          →      [lese].
       r(r(se))              →      [se].
       v3(v3(ztratilo))      →      [ztratilo].
       p6(p6(v))             →      [v].
```

/*Cvičení 8:
Uprav `g2.pl` tak, aby byla schopna postihnout shodu mezi jmennou skupinou `np` v hlavní větě a odpovídající větou vztažnou. Jde tu o spojení typu
Jeho žena, která *nenávidí auta, vášnivě miluje pletení* nebo
Můj syn miluje auto, které *nenávidí jeho žena.* */

/*Řešení 8: */
Do `g3.pl` doplníme dvě pravidla pro věty vztažné: v prvním je vztažným zájmenem vyjádřen `Ag` (tj.`Agr`), ve druhém je vztažným zájmenem vyjádřen `Ob (Obr)`. Dále doplníme pravidla pro `Agr` a `Obr` a ověříme si, že nám nechybí pravidla pro `np`. Nesmíme také zapomenout na pravidla pro `pr`, tj. pro vztažná zájmena. Pokud chceme, aby se v příslušných českých větách počítalo s obvyklou interpunkcí, musíme pro ně mít označení `Fm1` (čárka) a `Fm2` (čárka nebo tečka) a také samostatná pravidla. V jednodušší verzi je lze vynechat (pozor, na dvou místech).

```
ag(ag(Np),3,K,Nu,G) → np(Np,1,Nu,G).
ob(op(Np)) → np(Np,4,Nu,G).

sr(sr(Fm1,Agr,V1,Ob,Fm2),Nu,_) → fm(Fm1), agr(Agr,1,Nu,G), v1(V1,P,K,Nu,G),
sr(sr(Fm1,Obr,V1,Ag,Fm2),Nu,_) → fm(Fm1), obr(Obr,4,Nu,G), v1(V1,P,K,Nu,G),

agr(agr(Pr),3,K,Nu,G) → pr(Pr,1,Nu,G).
obr(obr(Pr)) → pr(Pr,4,Nu,G).


np(np(Pos,N,Sr),K,Nu,G) → n(Pos,N,K,Nu,G),sr(Sr,Nu,G).
np(np(N,Sr),K,Nu,G) → n(N,K,Nu,G),sr(Sr,Nu,G).
np(np(N),K,Nu,G) → n(N,K,Nu,G).


pr(pr(ktere),4,sg,n) → [ktere].
pr(pr(ktera),1,sg,f) → [ktera].


fm1(fm1(',')) → [','].
fm2(fm2(',')) → [','].
fm2(fm2('.')) → ['.'].
```

BIBLIOGRAFIE

BÉMOVÁ, A., KRÁLÍKOVÁ, K., K otázkám automatického zpracování českého tvarosloví, In: SaS, 49(1988), s.285nn.

BERKA, K., MLEZIVA, M., Co je logika, Praha, NPL, 1962.

BOGURAEV, B., BRISCOE, T., eds., Computational Lexicography for Natural Language Processing, Longman, London, New York, 1989.

BUZÁSSYOVÁ, K., Sémantická štruktúra slovenských deverbatív, Veda, Bratislava 1974.

CLOCKSIN, W. F., MELLISH, C. S., Programování v PROLOGU, český překlad, ZAK Slušovice, 1986.

ČEŠKA, M., RÁBOVÁ, Z., Gramatiky a jazyky, skriptum, VUT Brno, 1985, s.9-28.

DAHL, V., ABRAMSON, H., On Gapping Grammars, in: *Proceedings of the Second Int. Conf. on Logic Programming*, pp.77-88, Uppsala, Sweden, July 1984.

DOKULIL, M., DANEŠ, F., KUCHAŘ, J., Tvoření slov v češtině, Academia, Praha 1967.

ERHART, A, Základy jazykovědy, SPN, Praha, 1984.

FILLMORE, C., J., The Case for Case, in: Bach, E. and Harms, R., T., eds., *Universals in Linguistic Theory*, Holt, Rinehart and Winston, New York, pp. 1-88, 1968.

GAZDAR, G., Phrase Structure Grammar, in: The Nature of Syntactic Representations, ed. by G. K. Pullum and R. Jacobson, Dordrecht, 1982.

GAZDAR, G., MELLISH C., S., Natural Language Processing in PROLOG, Addison-Wesley, Wokingham, England, 1989.

GREPL, M., KARLÍK, P., Skladba spisovné češtiny, SPN, Praha, 1986.

HALASOVÁ, K., PALA, K., FRANC, S., Česká morfologie a syntax v PROLOGU, SOFSEM' 87, VUSEIAR Bratislava 1987.

HALASOVÁ, K., Algoritmický popis české formální morfologie substantiv a adjektiv, rkp. pro SPFFBU 1988.

HAVRÁNEK, B., JEDLIČKA, A., Česká mluvnice, SPN, Praha 1981.

CHOMSKY, N., Syntaktické struktury, Academia, Praha, 1966 (český překlad originálu publikovaného v r. 1957).

CHOMSKY, N., MILLER, G. A., Introduction to Formal Analysis of Natural Language, Formal Grammars, Finitary Models of Language Users, in: Handbook of Mathematical Psychology, vol. II, Wiley and Sons, New York, 1963, s.270–382.

KARLÍK, P., SVOBODA, A., Skladba češtiny pro cizince, skriptum Letní školy slovanských studií, Brno 1985.

KOMÁREK, M., Ke dvěma koncepcím stavby jednoduchých slovesných tvarů v češtině, Acta Universitatis Palackianaea Olomucensis, Studia Bohemica IV, SPN, Praha 1987.

KONEČNÁ, D., Algoritmický popis českých slovesných tvarů [Kandidátská disertace]. Praha 1964.

MATERNA, P., PALA, K., ZLATUŠKA, J., Logická analýza přirozeného jazyka, ACADEMIA, Praha, 1989.

NOVOTNÝ, M., S algebrou od jazyka ke gramatice a zpět, Academia, Praha, 1988.

OSOLSOBĚ, K., FRANC, S., PALA, K., An Algorithmic Dictionary of Czech for the IBM PC (and compatibles), výzk. zpráva, FF MU, Brno, 1990.

PALA, K., O procedurální gramatice pro češtinu, Sborník prací FF BU, A 30, 1982, s.103–122.

PETR, J., kol., Mluvnice češtiny 2, 3, Academia, Praha 1986, 1987.

PEREIRA, F., Logic for Natural Language Analysis, SRI, Technical Note, no 275, Stanford, 1983.

ROMPORTL, S., Návrh principu automatického šifrování a dešifrace gramatických příznaků českého slovesa při překládání z češtiny do češtiny. In: SbVUT, Brno 1961.

SGALL, P., Generativní popis jazyka a česká deklinace. ČSAV, Praha 1967.

SGALL, P., Das System der Kasusendungen im Tschechischen, The Prague Bulletin of Mathematical Linguistics 39, Praha 1983.

SGALL, P. et al., The Meaning of the Sentence and its Semantic and Pragmatic Aspects, Praha, Academia, 1986.

SLAVÍČKOVÁ, E., Retrográdní morfematický slovník češtiny, Academia, Praha 1975.

ŠMILAUER, V., Novočeská skladba, Praha, 1966.

ŠMILAUER, V., Učebnice větného rozboru, skriptum FF UK, Praha, 1967.

SOMERS, H. L., Valency and Case in Computational Linguistics, Edinburgh University Press, 1987.

TĚŠITELOVÁ, M., PETR, J., KRÁLÍK, J., Retrográdní slovník současné češtiny, Academia, Praha 1986.

UČEBNICE JAZYKA ČESKÉHO PRO 10. ROČ., SPN, Praha, 1954.

WELLS, R., Immediate Constituents, Language, Vol. 23, pp.81-117, 1947.

K. Pala, K. Osolsobě

Základy počítačové lingvistiky

(Skriptum filozofické fakulty MU)

Vedoucí ústavu: Radoslav Večerka
Vydavatel: Masarykova univerzita
Brno 1991
AA - 6,53, VA - 6,68
Cena 20,- Kčs
Tém. sk. 02/58
Vyd. číslo: 55-953A-91
ISBN 80-210-0341-3
2. přepracované vydání
Náklad 200 výtisků
Vytisklo ofsetem technické středisko FF MU