Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Toni Kocjan

Vgradnja objektno usmerjenih gradnikov v programski jezik PINS

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Bištjan Slivnik

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Na tem mestu zapišite, komu se zahvaljujete za izdelavo diplomske naloge. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.

Svoji dragi Alenčici.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CA	classification accuracy	klasifikacijska točnost
DBMS	database management system	sistem za upravljanje podatkovnih baz
SVM	support vector machine	metoda podpornih vektorjev

Povzetek

Naslov: Vgradnja objektno usmerjenih gradnikov v programski jezik PINS

Avtor: Toni Kocjan

V diplomskem delu bom predstavil programski jezik Atheris, ki je nastal kot nadgradnja programskega jezika PINS. Prog. jezik PINS, oz. prevajalnik zanj, je bil zgrajen tekom semestra pri predmetu prevajalniki in navidezni stroji. Ker mi je bilo delo na prevajalniku izjemno zanimivo, sem se odločil, da ustvarim svoj programski jezik in sam določim pravila zanj.

V diplomskem delu na kratko predstavim prevajalnike in programske jezike, kaj sploh so in kaj je njihov namen. Opišem kakšne so sodobne prakse pri ravoju prevajalnikov, s kakšnimi probleme se prevajalnik sooča ter kako je zgrajen.

Podrobneje bom obrazložil nadgradnje jezika PINS, s kakšnimi problemi sem se tekom razvoja soočal ter kako sem jih reševal. Pokazal bom nekaj primerov programov v mojem jeziku ter primerjal z drugimi jeziki.

Ključne besede: prevajalnik, programski jezik, sintaksa, semantika, Java, Swift.

Abstract

Title: Diploma thesis sample

Author: Toni Kocjan

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: compiler, programming language, syntax, semantics, Java, Swift.

Poglavje 1

Uvod

Razvoj prevajalnikov, ter s tem tudi programskih jezikov, je, po mojem mnenju, izjemno pomembna panoga v računalništvu. Programski jezik je medij, preko katerega komuniciramo z računalnikom. Prevajalniki razvijalcem omogočajo, da se med razvojem programske opreme ne rabijo osredotočati na nizkovojske detajle, ampak se lahko posvetijo reševanju praktičnih problemov. Naloga prevajalnika je, da pretvori človeku berljivo kodo v računalniku razumljivo zaporedje strojnih ukazov.

Dandanes lahko za razvoj programske opreme izbiramo med veliko količino programskih jezikov. Trenutno eni izmed najbolj popularnih so JavaScript, Java, Python in C++, popularnost pa dobivajo tudi novejši jeziki, kot so GoLang, Swift, Kotlin in podobni. [1]

S prevajalniki sem se prvič spoznal pri predmetu Prevajalniki in Navidezni Stroji (PINS) v drugem letnik na Fakulteti za Računalništvo in Informatiko. Tekom diplomske naloge bom predstavil programski jezik Atheris ter prevajalnik zanj. Prevajalnik je nastal kot nagradnja prevajalnika jezika PINS in obsega predvsem vgradnjo objektno usmerjenih gradnikov ter popolnoma spremenjeno sintakso.

Poglavje 2

Prevajalniki

2.1 Uvod v prevajalnike in programske jezike

Programski jezik je poseben jezik, ki se uporabljaja za razvoj programske opreme. Programski sistemi, ki poskrbijo, da se izvede pretvorba kode, napisane v programskem jeziku, v računalniku razumljivo obliko, se imenujejo *prevajalniki*.
Nekaj definicij:

1. **Računski model (angl. computational model):** zbirka vrednosti in računskih operacij
2. **Izračun (angl. computation)**: zaporodje operacij nad vrednostjo (ali več vrednosti), ki vrne nek rezultat
3. **Program:** specifikacija izračuna
4. **Programski jezik:** zapis (notacija) za pisanje programov

[2]

Program lahko predstavimo kot funkcijo, pri kateri je rezultat (angl. *output*) funkcija vhodnih parametrov (angl. *input*):

```
rezultat = program(vhodni parametri)
```

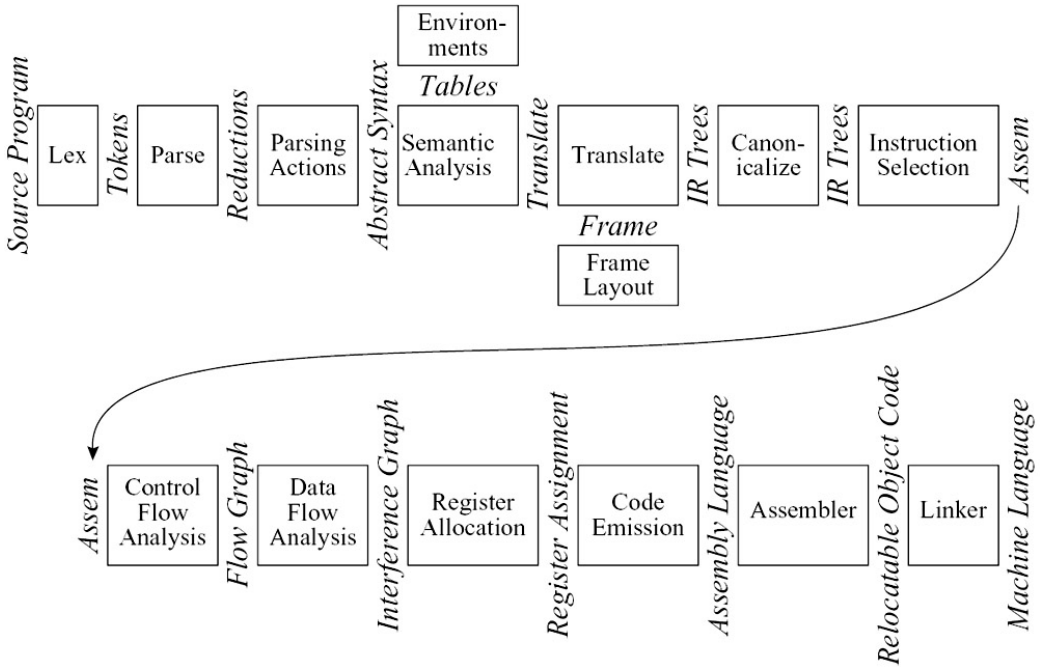
Iz drugega zornega kota si lahko program predstavljamo tudi kot model problemske domene, kjer je instanca izvedbe programa simulacija problema:

```
program = model problemske domene
izvedba programa = simulacija problema
```

[2]

2.2 Zgradba prevajalnika

Sodobni prevajalniki so pogosto organizirani v več posameznih faz, vsaka izmed njih pa operira na različnem nivoju abstrakcije jezika. [3]



Slika 2.1: Faze prevajalnika ter vmesniki, ki jih povezujejo med seboj.

Prevajalnik, da lahko program prevede iz ene oblike v drugo, mora program najprej analizirati, razumeti njegovo strukturo ter pomen, ter ga nato sestaviti nazaj v drugačno obliko.
Analizo programa običajno delimo v naslednje korake:

vrsta žetona	angl.	primeri
ime	identifier	x foo bar thisIsAnIdentifier
rezervirana beseda	keyword	while for if public override
operator	operator	, . && = ==
niz znakov	string	"this is a string"
znak	character	'a' 'x' '@'
celo št.	integer	10 125 082
decimalno št.	real	201.5 3.14 1.2e10

Tabela 2.1: Primeri žetonov v programskem jeziku Java

1. **Leksikalna analiza** (angl. *lexical analysis*)
2. **Sintaksna analiza** (angl. *syntax analysis*)
3. **Semantična analiza** (angl. *semantic analysis*)

[3]

2.2.1 Leksikalna analiza

Tako imenovani leksikalni analizator (modul, ki izvede leksikalno analizo) kot vhod prejme tok znakov (angl. *stream of characters*), kot izhod pa vrne tok v naprej definiranih žetonov (angl. *stream of tokens*). Žeton je običajno zgrajen iz imena, vrednosti (t.i. *lexeme*) ter lokacije v izvorni datoteki. [3]

Leksikalni žetoni:

Žeton (ali simbol) je zaporedje znakov, ki ga interpretiramo kot samostojno enoto v slovnici programskega jezika. [3]
Tabela 2.1 prikazuje nekaj vrst simbolov ter primere.

```
let x: Int
let y: Int
x * y
```

Listing 2.1: Primer programa v programskem jeziku Atheris

Rezultat:

```
[1:1-1:4]       LET:let
[1:5-1:6]       IDENTIFIER:x
[1:6-1:7]       COLON::
[1:8-1:11]      IDENTIFIER:Int
[1:11-1:12]     NEWLINE:\n
[2:1-2:4]       LET:let
[2:5-2:6]       IDENTIFIER:y
[2:6-2:7]       COLON::
[2:8-2:11]      IDENTIFIER:Int
[2:11-2:12]     NEWLINE:\n
[3:1-3:2]       IDENTIFIER:x
[3:3-3:4]       MUL:*
[3:5-3:6]       IDENTIFIER:y
EOF:$
```

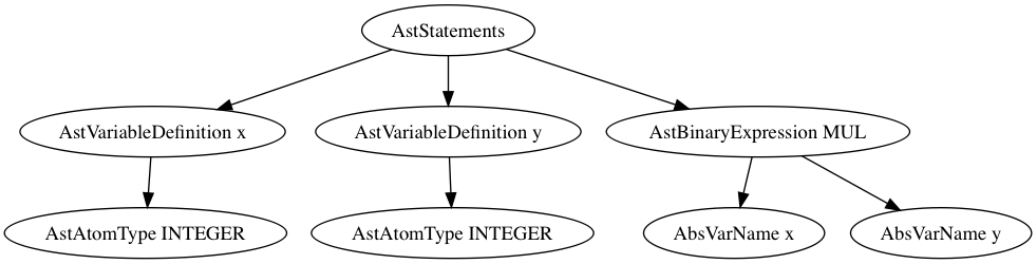
Listing 2.2: Rezultat leksikalne analize za program 2.1

2.2.2 Sintaksna analiza

Druga faza prevajanja je sintaksna analiza (angl. *syntax analysis* ali *parsing*). Naloga te faze je, da zagotovi, da je napisan program slovnično pravilen in v skladu s sintaksnimi pravili. Sintaksni analizator prejme kot vhod tok žetonov, ki ga zgenerira prejšnja faza, rezultat pa je abstraktno sintaksno drevo.

Abstraktno sintaksno drevo (AST) je drevesna podatkovna struktura, ki predstavlja slovnično strukturo programa. Vsako vozlišče drevesa pona-

zarja konstrukt v programski kodi.



Slika 2.2: Abstraktno sintaksno drevo za program 2.1.

Iz slike 2.2 lahko razberemo, da gre za dve definiciji spremenljivk in množenje.

Abstraktno sintaksno drevo je bistvenega pomena, saj nadaljne faze operirajo izključno nad njim.

2.2.3 Semantična analiza

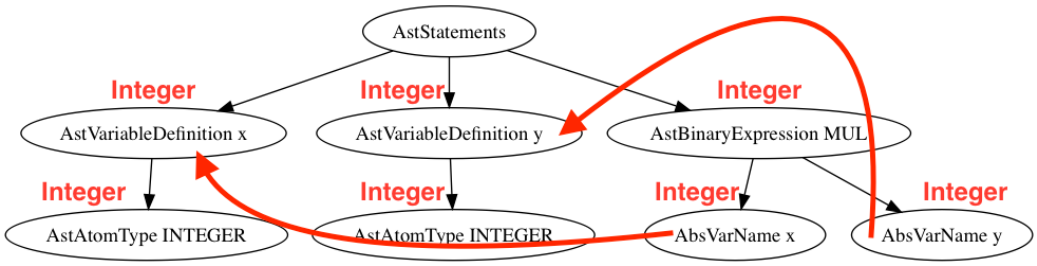
Semantična analiza poveže definicije spremenljivk z njihovimi uporabami ter preveri, ali so vsi izrazi pravilnih podatkovnih tipov. [3]

Običajno semantično analizo razdelimo na dve pod-fazi:

1. **Razreševanje imen:** zagotovi, da za vsako uporabo imena obstaja znotraj trenutnega območja vidnosti definicija z istim imenom, ter uporabo poveže z definicijo

2. **Preverjanje tipov:** vsakemu vozlišču v AST določi podatkovni tip, ter na podlagi postavljenih semantičnih pravil zagotovi, da so vsi izrazi pravilnih tipov

Pri implementaciji semantične analize nam pomaga *simbolna tabela*.



Slika 2.3: Rezultat semantične analize za program 2.1

Simbolna tabela

Simbolna tabela je podatkovna struktura, ki mapira imena v njihove definicije in podatkovne tipe. [3] Ker običajno programi vsebujejo več tisoč unikatnih definicij imen, mora podatkovna struktura omogočati učinkovito poizvedovanje. Iz slike 2.3 lahko razberemo, kaj se med izvajanjem semantične analize zgodi v ozadju: puščice predstavljajo povezave med definicijami in uporabami, evaluacija podatkovnih tipov pa je pri vseh vozliščih *Integer*, razen pri korenu, ki nima tipa oz. je tipa *Void*.

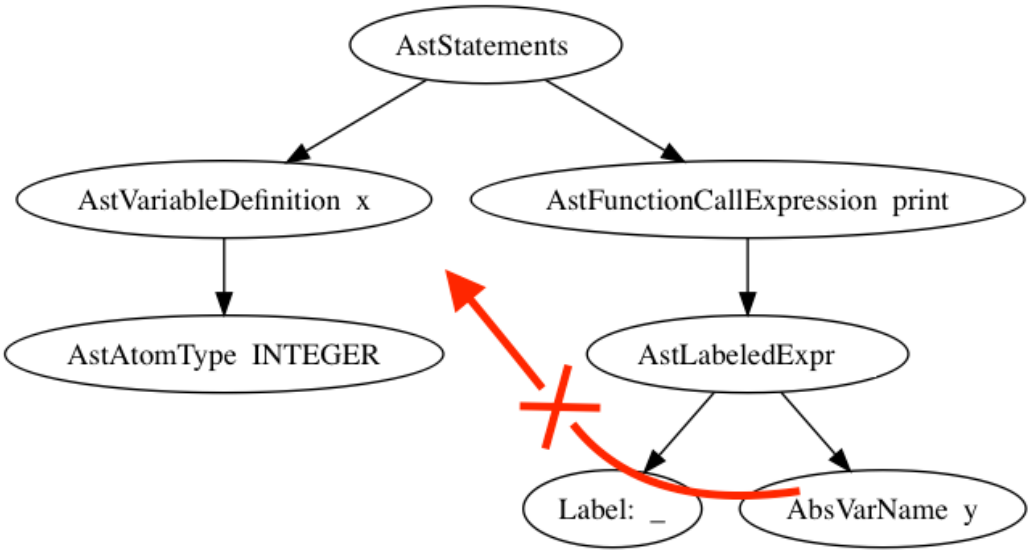
Kot sem omenil, semantična analiza zagotovi, da za vsako uporabo imena obstaja njena definicija, in da so podatkovni tipi pravilni. Sledita dva primera, kjer to ne drži:

```
let x: Int
print(y)
```

Listing 2.3: Primer programa, kjer spremenljivka y ni definirana

```
let x: Int
let s: String
x + s
```

Listing 2.4: Primer programa, kjer je napaka v podatkovnih tipih



Slika 2.4: Napaka v programu 2.3. Spremenljivka y ni definirana.

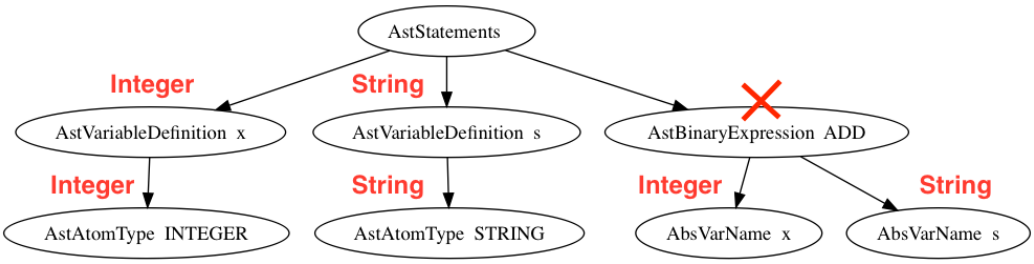
2.2.4 Klicni zapisi

V skoraj vsakem modernem programskem jeziku ima lahko funkcija *lokalne* spremenljivke, ki so kreirane ob vstopu v funkcijo. Hkrati lahko naenkrat obstaja več zapisov iste funkcije, zato je pomembno, da ima vsak zapis lastne instance lokalnih spremenljivk. [3]

V funkciji

```
let x: Int
let y: Int
x * y
```

se za vsak njen klic ustvari nova instanca x, ki jo inicializira klicalec funkcije. Ker je funkcija rekurzivna, živi v pomnilniku naenkrat veliko x-ev. Podobno se ob vsakem vstopu v jedro funkcije ustvari tudi nova instanca y. [3]



Slika 2.5: Napaka v programu 2.4. Seštevanje med podatkovnima tipoma *Integer* in *String* ni dovoljeno.

Sklad

Klicni zapisi funkcij se shranjujejo na sklad. Sklad je v pomnilniku predstavljen kot velika tabela s posebnik registrimo imenovanim *stack pointer* oz. kazalec na sklad, ki kaže na konec sklada. Ob vsakem klicu funkcije se sklad poveča za velikost klicnega zapisa klicane funkcije. Podobno se ob vrnitvi funkcije zmanjša za enako vrednost. Prostor v klicnem zapisu namenjen hrambi vhodnih parametrov, lokalnih spremenljivk in ostalih registrov se imenuje *activation record*. [3]

Kazalec na klicni zapis

Predpostavimo da funkcija g kliče funkcijo f. Ob vstopu v f kazalec na sklad (SP) kaže na prvi vhodni argument funkciji f. Nov prostor na skladu je rezerviran tako, da se od SP odšteje velikost klicnega zapisa f. Tako SP sedaj kaže na konec sklada. Stara vrednost SP postane nova vrednost kazalca na klicni zapis (*angl. frame pointer*). V programskih jezikih, kjer je velikost klicnega zapisa za posamezno funkcijo konstantna, je vrednost kazalca na klicni (FP) zapis vedno izračunljiva, zato si je ni potrebno posebaj shranjevati na sklad. FP = SP + velikost sklada. [3]

Prenos parametrov

Standarden način klicanja funkcij je, da klicoča funkcija rezervira prostor na skladu za prenosih njenih izhodnih parametrov (oz. vhodnih parametrov

za klicano funkcijo). [3]

Prostor za njih se običajno nahaja pred SP. Klicana funkcija tako naslov njenega i-tega parametra izračuna z enačbo

```
argumentAddress(i) = FP + (4 * i)
```

Lokalne spremenljivke

Prevajalnik mora na skladu alocirati dovolj prostora za vse lokalne spremenljivke.

Static link

V jezikih, ki podpirajo gnezdenje funkcij, lahko gnezdene funkcije dostopajo do spremenljivk, ki se nahajajo v zunanjih funkcijah. Da lahko gnezdena funkcija dostopa do spremenljivk, ki niso na njenem skladu, ji ob klicu poleg ostalih parametrov posredujemo FP funkcije, ki jo neposredno definira. Temu kazalcu rečemo *static link*.

```
func f() {
    var x: Int = 10

    func e() {
        var z: Int = 100
    }

    func g() {
        var y: Int = 20
        func h() {
            print(y, x)
        }
    }
}
```

Listing 2.5: Primer gnezdenih funkcij

Primer 2.5 vsebuje dve gnezdeni funkciji g in h. Funkcija h lahko dostopa do vseh spremenljivk, ki so definirane v

Poglavje 3

Matematično okolje in sklicevanje na besedilne konstrukte

Matematična ali popolna indukcija je eno prvih orodij, ki jih spoznamo za dokazovanje trditev pri matematičnih predmetih.

Izrek 3.1 *Za vsako naravno število n velja*

$$n < 2^n. \tag{3.1}$$

Dokaz. Dokazovanje z indukcijo zahteva, da neenakost (3.1) najprej preverimo za najmanjše naravno število – 0. Res, ker je $0 < 1 = 2^0$, je neenačba (3.1) za $n = 0$ izpolnjena.

Sledi indukcijski korak. S predpostavko, da je neenakost (3.1) veljavna pri nekem naravnem številu n, je potrebno pokazati, da je ista neenakost v veljavi tudi pri njegovem nasledniku – naravnem številu $n + 1$. Računajmo.

$$\begin{aligned} n + 1 &< 2^n + 1 && (3.2)\\ &\leq 2^n + 2^n && (3.3)\\ &= 2^{n+1} \end{aligned}$$

Neenakost (3.2) je posledica indukcijske predpostavke, neenakost (3.3) pa enostavno dejstvo, da je za vsako naravno število n izraz 2^n vsaj tako velik kot 1. S tem je dokaz Izreka 3.1 zaključen. □

Opazimo, da je LaTeX številko izreka podredil številki poglavja. Na podoben način se lahko sklicujemo tudi na druge besedilne konstrukte, kot so med drugim poglavja, podpoglavja in plovke, ki jih bomo spoznali v naslednjem poglavju.

Poglavje 4

Plovke: slike in tabele

Slike in daljše tabele praviloma vključujemo v dokument kot plovke. Pozicija plovke v končnem izdelku ni pogojena s tekom besedila, temveč z izgledom strani. LaTeX bo skušal plovko postaviti samostojno, praviloma na mestu, kjer se pojavi v izvornem besedilu, sicer pa na vrhu strani, na kateri se na takšno plovko prvič sklicujemo. Pri tem pa bo na vsako stran končnega izdelka želel postaviti tudi sorazmerno velik del besedila. V skrajnem primeru, če imamo res preveč plovk na enem mestu besedila, ali če je plovka previsoka, se bo LaTeX odločil za stran popolnoma zapolnjeno s plovkami.

Poleg tega, da na položaj plovke vplivamo s tem, kam jo umestimo v izvorno besedilo, lahko na položaj plovke na posamezni strani prevedenega besedila dodatno vplivamo z opcijami `here, top` in `bottom`. Zelo velike slike je najbolje postaviti na posebno stran z opcijo `page`. Skaliranje slik po njihovi širini lahko prilagodimo širini strani tako, da kot enoto za dolžino uporabimo kar širino strani, npr. `0.5\textwidth` bo raztegnilo sliko na polovico širine strani.

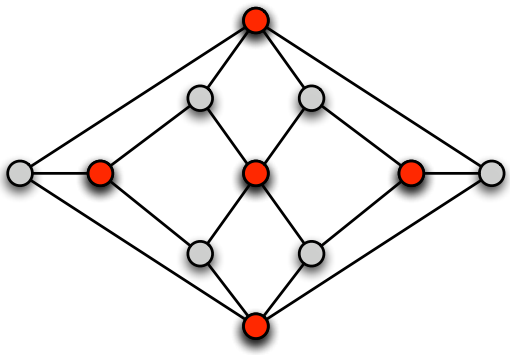
Na vse plovke se moramo v besedilu sklicevati, saj kot beseda pove, plovke plujejo po besedilu in se ne pojavijo točno tam, kjer nastopajo v izvornem besedilu. Sklic na plovko v besedilu in sama plovka naj bosta čimbližje skupaj, tako da bralcu ne bo potrebno listati po diplomi. Upoštevajte pa, da se naloge tiska dvostransko in da se hkrati vidi dve strani v dokumentu!

Na to, kje se bo slika ali druga plovka pojavila v postavljenem besedilu torej najbolj vplivamo tako, da v izvorni kodi plovko premikamo po besedilu nazaj ali naprej!

Tabele ja najbolje oblikovati kar neposredno v LaTeXu, saj za oblikovanje tabel obstaja zelo fleksibilno okolje `tabular` (glej tabelo 4.1). Slike po drugi strani pa je bolje oblikovati oziroma izdelati z drugimi orodji in programi in se v LaTeXu le sklicevati na ustrezno slikovno datoteko.

4.1 Formati slik

Bitne slike, vektorske slike, kakršnekoli slike, z LaTeXom lahko vključimo vse. Slika 4.1 je v `.pdf` formatu. Pa res lahko vključimo slike katerihkoli forma-



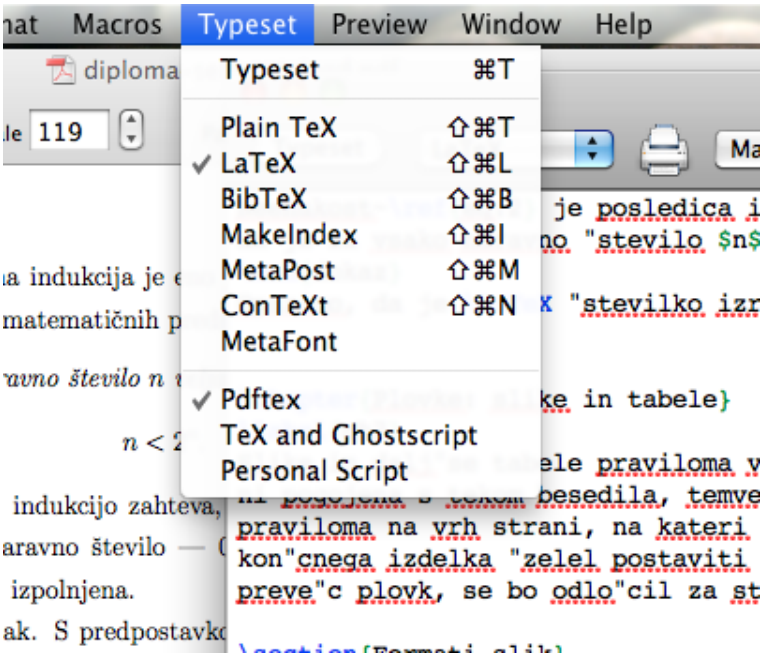
Slika 4.1: Herschelov graf, vektorska grafika.

tov? Žal ne. Programski paket LaTeX lahko uporabljamo v več dialektih. Ukaz `latex` ne mara vključenih slik v formatu Portable Document Format `.pdf`, ukaz `pdflatex` pa ne prebavi slik v Encapsulated Postscript Formatu `.eps`. Strnjeno je vključevanje različnih vrst slikovnih datotek prikazano v tabeli 4.1.

Nasvet? Odločite se za uporabo ukaza `pdflatex`. Vaš izdelek bo brez vmesnih stopenj na voljo v .pdf formatu in ga lahko odnesete v vsako tiskarno.

ukaz/format	`.pdf`	`.eps`	ostali formati
`pdflatex`	da	ne	da
`latex`	ne	da	da

Tabela 4.1:



Slika 4.2: Kateri dialekt uporabljati?

Če morate na vsak način vključiti sliko, ki jo imate v `.eps` formatu, jo vnaprej pretvorite v alternativni format, denimo `.pdf`.

Včasih se da v okolju za uporabo programskega paketa LaTeX nastaviti na kakšen način bomo prebavljali vhodne dokumente. Spustni meni na Sliki **??** odkriva uporabo LaTeXa v njegovi pdf inkarnaciji — `pdflatex`. Vključena slika 4.2 je seveda bitna.

Na vse tabele se moramo v besedilu, podobno kot na slike, tudi sklicevati, saj kot plovke v oblikovanem besedilo niso nujno na istem mestu kot v izvornem besedilu.

4.1.1 Podnapisi k slikam in tabelam

Vsaki sliki ali tabeli moramo dodati podnapis, ki na kratko pojasnuje, kaj je na sliki ali tabeli. Če nekdo le prelista diplomsko delo, naj bi že iz slik in njihovih podnapisov lahko na grobo razbral, kakšno temo naloga obravnava.

Če slike povzamemo iz drugih virov, potem se moramo v podnapisu k taki sliki sklicevati na ta vir!

Poglavje 5

Struktura strokovnih besedil

Strokovna besedila imajo ustaljeno strukturo, da bi lahko hitreje in lažje brali in predvsem razumeli taka besedila, saj načeloma vemo vnaprej, kje v besedilu se naj bi nahajale določene informacije.

Najbolj osnovna struktura strokovnega besedila je:

naslov besedila, ki naj bo sicer kratek, a kljub temu dovolj poveden o vsebini besedila,

imena avtorjev so običajno navedena po teži prispevka, prvi avtor je tisti, ki je besedilo dejansko pisal, zadnji pa tisti, ki je raziskavo vodil,

kontaktni podatki – poleg imena in naslova institucije je potreben vsaj naslov elektronske pošte,

povzetek je kratko besedilo, ki povsem samostojno povzame vsebino in izpostavi predvsem glavne rezultate ali zaključke,

ključne besede so tudi namenjene iskanju vsebin med množico člankov,

uvodno poglavje uvede bralca v tematiko besedila, razloži kaj je namen besedila, predstavi področje o katerem besedilo piše (če temu ni namenjeno v celoti posebno poglavje) ter na kratko predstavi strukturo celotnega besedila,

poglavja tvorijo zaokrožene celote, ki se po potrebi še nadalje členijo na podpoglavja, namenjena so recimo opisu orodij, ki smo jih uporabili pri delu, teoretičnim rezultatom ali predstavitvi rezultatov, ki smo jih dosegli,

zaključek še enkrat izpostavi glavne rezultate ali ugotovitve, jih primerja z dosedanjimi in morebiti poda tudi ideje za nadaljne delo,

literatura je seznam vseh virov, na katere smo se pri svojem delu opirali, oziroma smo se na njih sklicevali v svojem besedilu.

Strokovna besedila običajno pišemo v prvi osebi množine, v nevtralnem in umirjenem tonu. Uporaba sopomenk ni zaželjena, saj želimo zaradi lažjega razumevanja za iste pojme vseskozi uporabljati iste besede. Najpomembnejše ugotovitve je smiselno večkrat zapisati, na primer v povzetku, uvodu, glavnem delu in zaključku. Vse trditve naj bi temeljile bodisi na lastnih ugotovitvah (izpeljavah, preizkusih, testiranjih) ali pa z navajanjem ustreznih virov.

Največ se lahko naučimo s skrbnim branjem dobrih zgledov takih besedil.

Poglavje 6

Pogoste napake pri pisanju v slovenščini

V slovenščini moramo paziti pri uporabi pridevnikov, ki se ne sklanjajo kot so npr. kratice. Pravilno pišemo model CAD in **ne** CAD model!

Pri sklanjanju tujih imen ne uporabljamo vezajev, pravilno je Applov operacijski sistem in **ne** Apple-ov.

Pika, klicaj in vprašaj so levostični: pred njimi ni presledka, za njimi pa. Klicajev in vprašajev se v strokovnih besedilih načeloma izogibamo. Oklepaji so desnostični in zaklepaji levostični (takole).

V slovenščini pišemo narekovaje drugače kot v angleščini! Običajno uporabljamo dvojne spodnje-zgornje narekovaje: „slovenski narekovaji“. Za slovenske narekovaje je v tej LaTeXovi predlogi definiran nov ukaz `\sn{ ... }`.

Vezaj je levo in desno stičen: `slovensko-angleški slovar` in ga pišemo z enim pomišljajem.

V slovenščini je pred in po pomišljaju presledek, ki ga v LaTeXu pišemo z dvema pomišljajema: `Pozor -- hud pes!` V angleščini pa je za razliko pomišljaj levo in desno stičen in se v LaTeXu piše s tremi pomišljaji: `---`. S stičnim pomišljajem pa lahko nadomeščamo predlog od ... do, denimo pri navajanju strani, npr. preberite strani 7–11 (`7--11`).

„Pred ki, ko, ker, da, če vejica skače“. To osnovnošolsko pravilo smo v

življenju po potrebi uporabljali, dopolnili, morda celo pozabili. Pravilo sicer drži, ampak samo če je izpolnjenih kar nekaj pogojev (npr. da so ti vezniki samostojni, enobesedni, ne gre za vrivek itd.). Povedki so med seboj ločeni z vejicami, razen če so zvezani z in, pa, ter, ne–ne, niti–niti, ali, bodisi, oziroma. Sicer pa je bolje pisati kratke stavke kot pretirano dolge.

V računalništvu se stalno pojavljajo novi pojmi in nove besede, za katere pogosto še ne obstajajo uveljavljeni slovenski izrazi. Kadar smo v dvomih, kateri slovenski izraz je primeren, si lahko pomagamo z Računalniškim slovarčkom [13].

Poglavje 7

Koristni nasveti pri pisanju v LaTeXu

Programski paket LaTeX je bil prvotno predstavljen v priročniku [7] in je v resnici nadgradnja sistema TeX avtorja Donalda Knutha [5], znanega po svojih knjigah o umetnosti programiranja, ter Knuth-Bendixovem algoritmu [6].

Različnih implementacij LaTeXa je cela vrsta. Za OS X priporočamo TeXShop, za Windows PC pa MikTeX. Spletna verzija, ki poenostavi sodelovanje pri pisanju, je npr. ShareLaTeX.

Včasih smo si pri pisanju v LaTeXu pomagali predvsem s tiskanimi priročniki, danes pa je enostavneje in hitreje, da ob vsakem problemu za pomoč enostavno povprašamo Google, saj je na spletu cela vrsta forumov za pomoč pri TeXiranju.

LaTeX včasih ne zna deliti slovenskih besed, ki vsebujejo črke s strešicami. Če taka beseda štrli preko desnega roba, LaTeXu pokažemo, kje lahko tako besedo deli, takole: `ra\-ču\-nal\-ni\-štvo`. Katere vrstice štrlijo preko desnega roba, se lahko prepričamo tako, da dokument prevedemo s vključeno opcijo `draft`: `\documentclass[a4paper, 12pt, draft]{book}`.

Predlagamo, da v izvornem besedilu začenjate vsak stavek v novi vrstici, saj LaTeX sam razporeja besede po vrsticah postavljenega besedila. Bo pa zato iskanje po izvornem besedilu in popravljanje veliko hitrejše. Večina

sistemov za TEXiranje sicer omogoča s klikanjem enostavno prestopanje iz prevedenega besedila na ustrezno mesto v izvornem besedilu in obratno.

Boljšo preglednost dosežemo, tako kot pri pisanju programske kode, tudi z izpuščanjem praznih vrstic za boljšo preglednost strukture izvornega besedila.

S pomočjo okolja `\begin{comment} ... \end{comment}` lahko hkrati zakomentiramo več vrstic izvornega besedila.

Pri spreminjanju in dodajanju izvornega besedila je najbolje pogosto prevajati, da se sproti prepričamo, če so naši nameni izpolnjeni pravilno.

Kadar besedilo, ki je že bilo napisano z nekim vizualnim urejevalnikom (npr. z Wordom), želimo prenesti v LATEX, je tudi najbolje to delati postopoma s posameznimi bloki besedila, tako da lahko morebitne napake hitro identificiramo in odpravimo. Za prevajanje Wordovih datotek v LATEX sicer obstajajo prevajalniki, ki pa običajno ne generirajo tako čisto logično strukturo besedila, kot jo LATEX omogoča. Hiter in enostaven način prevedbe besedila, ki zahteva sicer ročne dopolnitve, poteka tako, da besedilo urejeno z vizualnim urejevalnikom najprej shranimo v formatu pdf, nato pa to besedilo uvozimo v urejevalnik, kjer urejamo izvorno besedilo v formatu LATEX.

7.1 Pisave v LATEXu

V LATEXovem okolju lahko načeloma uporabljamo poljubne pisave. Izbira poljubne pisave pa ni tako enostavna kot v vizualnih urejevalnikih besedil. Posamezne oblikovno medseboj usklajene pisave so običajno združene v družine pisav. V LATEXu se privzeta družina pisav imenuje Computer Modern, kjer so poleg navadnih črk (roman v LATEXu) na voljo tudi kurzivne črke (*italic* v LATEXu), krepke (**bold** v LATEXu), kapitelke (SMALL CAPS v LATEXu), linearne črke (sans serif v LATEXu) in druge pisave. V istem dokumentu zaradi skladnega izleda uporabljamo običajno le pisave ene družine.

Ko začenjamo uporabljati LATEX, je zato najbolj smiselno uporabljati kar privzete pisave, s katerimi je napisan tudi ta dokument. Z ustreznimi ukazi

lahko nato preklapljamo med navadnimi, kurzivnimi, krepkimi in drugimi pisavami. Zelo enostavna je tudi izbira velikosti črk. LaTeX odlično podpira večjezičnost, tudi v sklopu istega dokumenta, saj obstajajo pisave za praktično vse jezike, tudi take, ki ne uporabljajo latinskih črk.

Za prikaz programske kode se pogosto uporablja pisava, kjer imajo vse črke enako širino, kot so črke na mehanskem pisalnem stroju (`typewriter` v LaTeXu).

Najbolj priročno okolje za pisanje kratkih izsekov programske kode je okolje `verbatim`, saj ta ohranja vizualno organizacijo izvornega besedila in ima privzeto pisavo pisalnega stroja.

```
for (i = 0; i < 100; i++)
   for (j = i; j < 10; j++)
      some_function(i, j);
```

Poglavje 8

Kaj pa literatura?

Kot smo omenili že v uvodu, je pravi način za citiranje literature uporaba BibTeXa [8]. BibTeX zagotovi, da nobene obvezne informacije pri določeni vrsti literature ne izpustimo in da vse informacije o določeni vrsti vira dosledno navajamo na enak način.

Osnovna ideja BibTeXa je, da vse informacije o literaturi zapisujemo v posebno datoteko, v našem primeru je to `literatura.bib`. Vsakemu viru v tej datoteki določimo simbolično ime. V našem primeru je v tej datoteki nekaj najbolj značilnih zvrsti literature, kot so knjige [7], članki v revijah [11] in zbornikih konferenc [10], spletni viri [8, 13, 12], tehnično poročilo [1], diplome [4] itd. Diploma [4] iz leta 1990 je bila prva diploma na Fakulteti za elektrotehniko in računalništvo, ki je bila oblikovana z LaTeXom!

Po vsaki spremembi pri sklicu na literaturo moramo najprej prevesti izvorno besedilo s prevajalnikom LaTeX, nato s prevajalnikom BibTeX, ki ustvari datoteko `vzorec_dip_Seminar.bbl`, in nato še dvakrat s prevajalnikom LaTeX.

Kako natančno se spisek literature nato izpiše (ali po vrstnem redu sklicevanja, ali po abecedi priimkov prvih avtorjev, ali se imena avtorjev pišejo pred priimki itd.) je odvisno od stilske datoteke. V diplomi bomo uporabili osnovno stilsko datoteko `plain`, ki vire razporedi po abecedi. Zato je potrebno pri določenih zvrsteh literature, ki nima avtorjev, dodati polje `key`,

ki določi vrstni red vira po abecedi.

Z uporabo BibTeXa v slovenščini je še nekaj nedoslednosti, saj so pomožne besede, ki jih BibTeX sam doda, kot so *editor*, *pages* in besedica *and* pred zadnjim avtorjem, če ima vir več avtorjev [1], zapisane v angleščini, čeprav smo izbrali opcijo `slovene` pri paketu `babel`. To nedoslednost je možno popraviti z ročnim urejanjem datoteke `vzorec_dip_Seminar.bbl`, kar pa je smiselno šele potem, ko bibliografije v datoteki `literatura.bib` ne bomo več spreminjali, oziroma ne bomo več dodajali novih sklicev na literaturo v izvornem besedilu. Vsebino datoteke `vzorec_dip_Seminar.bbl` lahko na koncu urejanja tudi vključimo kar v izvorno besedilo diplome, tako da je vso besedilo, vključno z literaturo, zajeto le v eni datoteki.

Ko začenjamo uporabljati BibTeX je lažje, če za urejanje datoteke .bib uporabljamo kar isti urejevalnik kot za urejanje datotek .tex, čeprav obstajajo tudi posebni urejevalniki oziroma programi za delo z BibTeXom.

Le če se bomo na določen vir v besedilu tudi sklicevali, se bo pojavil tudi v spisku literature. Tako je avtomatično zagotovljeno, da se na vsak vir v seznamu literature tudi sklicujemo v besedilu diplome. V datoteki `.bib` imamo sicer lahko veliko več virov za literaturo, kot jih bomo uporabili v diplomi.

Vire v formatu BibTeX lahko enostavno poiščemo in prekopiramo iz akademskih spletnih portalov za iskanje znanstvene literature v našo datoteko `.bib`, na primer v Google učenjaku. Izvoz v Google učenjaku še dodatno poenostavimo, če v nastavitvah izberemo BibTeX kot želeni format za izvoz navedb. Navedbe, ki jih na tak način prekopiramo, pa moramo pred uporabo vseeno preveriti, saj so taki navedki običajno generirani povsem avtomatično.

Pri sklicevanju na literaturo na koncu stavka moramo paziti, da je pika po ukazu `\cite{ }`. Da LaTeX ne bi delil vrstico ravno tako, da bi sklic na literaturo v oglatih oklepajih začel novo vrstico, lahko pred sklicem na literaturo dodamo nedeljiv presledek: `~\cite{ }`.

8.1 Izbiranje virov za spisek literature

Dandanes se skoraj vsi pri iskanju informacij vedno najprej lotimo iskanja preko svetovnega spleta. Rezultati takega iskanja pa so pogosto spletne strani, ki danes obstajajo, jutri pa jih morda ne bo več, ali pa vsaj ne v taki obliki, kot smo jo prebrali. Smisel navajanja literature pa je, da tudi po dolgih letih nekdo, ki bo bral vašo diplomo, lahko poišče vire, ki jih navajate v diplomi. Taki viri pa so predvsem članki v znanstvenih revijah, ki se arhivirajo v knjižnicah, založniki teh revij pa večinoma omogočajo tudi elektronski dostop do arhiva vseh njihovih člankov.

Znanstveni rezultati, ki so objavljeni v obliki recenziranih člankov, bodisi v konferenčnih zbornikih, še bolje pa v znanstvenih revijah, so veliko bolj izčiščen in zanesljiv vir informacij, saj so taki članki šli skozi recenzijski postopek. Zato na svetovnem spletu začenjamo iskati vire za strokovna besedila predvsem preko akademskih spletnih portalov, kot so npr. Google učenjak, Research Gate ali Academia, saj so na teh portalih rezultati iskanja le akademske publikacije. Če je za dostop do nekega članka potrebno plačati, se obrnemo za pomoč in dodatne informacije na našo knjižnico.

Če res ne gre drugače, pa je pomembno, da pri sklicevanju na spletni vir, vedno navedemo tudi datum, kdaj smo dostopali do tega vira.

Poglavje 9

Sistem STUDIS in PDF/A

Elektronsko verzijo diplome moramo oddati preko sistema STUDIS v formatu PDF/A [9]. Natančneje v formatu PDF/A-1b.

LaTeX in omenjeni format imata še nekaj težav s sobivanjem. Paket `pdfx.sty`, ki naj bi LaTeXu omogočal podporo formatu PDF/A ne deluje v skladu s pričakovanji. Ta predloga delno ustreza formatu, vsekakor dovolj, da jo študentski informacijski sistem sprejme. Znaten del rešitve je prispeval Damjan Cvetan.

V predlogi, poleg izvornega dokumenta `.tex` in vloženih slik `pic1.pdf` in `pic2.png`, potrebujemo še predlogo datoteke z metapodatki `pdfa-1b.xmp` in datoteko z barvnim profilom `sRGBIEC1966-2.1.icm`.

Poglavje 10

Sklepne ugotovitve

Uporaba LaTeXa in BibTeXa je v okviru Diplomskega seminarja **obvezna**! Izbira LaTeX ali ne LaTeX pri pisanju dejanske diplomske naloge pa je prepuščena dogovoru med vami in vašim mentorjem.

Res je, da so prvi koraki v LaTeXu težavni. Ta dokument naj vam služi kot začetna opora pri hoji. Pri kakršnihkoli nadaljnih vprašanjih ali napakah pa svetujem uporabo Googla, saj je spletnih strani za pomoč pri odpravljanju težav pri uporabi LaTeXa ogromno.

Literatura

[1] Michael Riis Andersen, Thomas Jensen, Pavel Lisouski, Anders Krogh Mortensen, Mikkel Kragh Hansen, Torben Gregersen, and Peter Ahrendt. Kinect depth sensor evaluation for computer vision applications. Technical report, Department of Engineering, Aarhus University, 2012.

[2] A complete description of a programming language includes the computational model, the syntax and semantics of programs, and the pragmatic considerations that shape the language. Dosegljivo: `http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/intro.htm`. [Dostopano: 26.11.2017].

[3] Andrew W. Appel and Jens Palsberg. *Modern Compiler Implementation in Java, Second Edition.* Cambridge University Press, 2002.

[4] Andreja Balon. Vizualizacija. Diplomska naloga, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, 1990.

[5] Donald knuth. Dosegljivo: `https://sl.wikipedia.org/wiki/Donald_Knuth`. [Dostopano: 1. 10. 2016].

[6] Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: Classical papers on computational logic 1957–1966*, pages 342–376. Springer, 1983.

[7] Leslie Lamport. *LaTEX: A Document Preparation System.* Addison-Wesley, 1986.

[8] Oren Patashnik. BibTeXing. Dosegljivo: `http://bibtexml.sourceforge.net/btxdoc.pdf`, 1988. [Dostopano 5. 6. 2016].

[9] PDF/A. Dosegljivo: `http://en.wikipedia.org/wiki/PDF/A`, 2005. [Dostopano: 5. 6. 2016].

[10] Peter Peer and Borut Batagelj. Art—a perfect testbed for computer vision related research. In *Recent Advances in Multimedia Signal Processing and Communications*, pages 611–629. Springer, 2009.

[11] Franc Solina. 15 seconds of fame. *Leonardo*, 37(2):105–110, 2004.

[12] Franc Solina. Light fountain—an interactive art installation. Dosegljivo: `https://youtu.be/CS6x-QwJywg`, 2015. [Dostopano: 9. 10. 2015].

[13] Matjaž Gams (ured.). DIS slovarček, slovar računalniških izrazov, verzija 2.1.70. Dosegljivo: `http://dis-slovarcek.ijs.si`. [Dostopano: 1. 10. 2016].