

# Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).



Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Science and Engineering



Master's Thesis

## **Framework Squander Usage**

*Bc. Martin Kožený*

Supervisor: Ing. Jiří Daněček

Study Programme: Electrical Engineering and Information Technology

Field of Study: Computer Science and Engineering

September 30, 2011



## Aknowledgements

At this point I would like to thank supervisor of my thesis Mr Ing. Jiří Daněček and author of Squander framework Mr Aleksandar Milicevics for valuable comments and advices to this work.



## Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague, 30. 9. 2011

.....





# Abstract

The aim of this work is to describe and study framework Squander developed on Massachusetts Institute of Technology by Mr. Aleksandar Milicevic. This framework brings into language Java another way of programming, which can improve effectiveness of implementing and computation performance of the program.

Work shows how was this framework used for implementing set of algorithms, especially NP-complete graph algorithms, and compares that implementation with common imperative way of programming.

# Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její podstatu. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.

Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Logical programming principles</b>	<b>3</b>
2.1	Logical programming paradigm . . . . .	3
2.1.1	Facts . . . . .	4
2.1.2	Rules . . . . .	5
2.2	Terminology . . . . .	6
<b>3</b>	<b>Meaning of annotations</b>	<b>7</b>
<b>4</b>	<b>Framework description</b>	<b>9</b>
<b>5</b>	<b>Implementation of algorithms</b>	<b>11</b>
<b>6</b>	<b>Comparison</b>	<b>13</b>
<b>7</b>	<b>Conclusion</b>	<b>15</b>
<b>A</b>	<b>Testování zaplnění stránky a odsazení odstavců</b>	<b>17</b>
<b>B</b>	<b>Pokyny a návody k formátování textu práce</b>	<b>21</b>
B.1	Vkládání obrázků . . . . .	21
B.2	Kreslení obrázků . . . . .	22
B.3	Tabulky . . . . .	22
B.4	Odkazy v textu . . . . .	23
B.4.1	Odkazy na literaturu . . . . .	23
B.4.2	Odkazy na obrázky, tabulky a kapitoly . . . . .	25
B.5	Rovnice, centrovaná, číslovaná matematika . . . . .	25
B.6	Kódy programu . . . . .	26
B.7	Další poznámky . . . . .	26
B.7.1	České uvozovky . . . . .	26
<b>C</b>	<b>Seznam použitých zkratk</b>	<b>27</b>
<b>D</b>	<b>UML diagramy</b>	<b>29</b>
<b>E</b>	<b>Instalační a uživatelská příručka</b>	<b>31</b>

<b>F</b>	<b>Obsah přiloženého CD</b>
----------	-----------------------------

<b>33</b>
-----------

# List of Figures

B.1	Popiska obrázku . . . . .	22
F.1	Seznam přiloženého CD — příklad . . . . .	33



# List of Tables

B.1 Ukázka tabulky . . . . .	22
------------------------------	----





# Chapter 1

## Introduction

As was said in abstract, the main purpose of this thesis is to describe and use framework Squander developed at Massachusetts Institute of Technology (MIT) by Mr Aleksandar Milicevics. This framework brings to Java declarative constructs, which are useful for implementing programs that involve computations that are relatively easy to specify but hard to solve algorithmically. In such cases is better to use declarative constraints to naturally express the core of the computation, whereas imperative code is natural choice to read input parameters and setting up data structures for the computation. This is big advantage of this framework, that programmer can smoothly switch between declarative logical and imperative programming.

By ability of mixing imperative and declarative code can programmer easily express constraints of problem in terms of existing data structures and objects on the heap. Despite having overhead of encoding and decoding, it is surprising how is competitive Squander's SAT-based solution with specialized heuristic developed for concrete problem.



## Chapter 2

# Logical programming principles

The most known logical programming in the world is definitely Prolog and thereby we can describe properties of logical programming using this language. His name is derived from term PROgramming in LOGic and was developed for programming of symbolic computation. His success led to formation new discipline in mathematical information technology - **logical programming**.

Logical programming focus on description of relation's properties without need to know how to do that.

## 2.1 Logical programming paradigm

Logical programming differ from imperative languages in following points:

1. no assignment statement
2. no cycles, no branching
3. no flow control
4. object is marked as *variable*, which satisfies some set of conditions, that are being during computation more specified

Logical programming is based on following concepts:

1. declaring facts about objects and relations between objects
2. declaring valid rules about objects and relations between themselves and computing queries

In logical programming are *facts* unconditional commands and *rules* are conditional commands. Facts and rules are stored in one shared database. Language does not differ between program and data.

### 2.1.1 Facts

For expressing *facts* and *rules* are used *clauses*. Facts are used for expressing unconditionally true assertions and are clauses with defined headers, but with no body. Usual way to illustrate how to composed facts are family relationships.

```
parent(john, jane).
parent(ann, jane).
parent(john, richard).
parent(ann, richard).
man(john).
man(richard).
womam(ann).
woman(jane).
```

Every clause declares concrete fact about relation. We can see that relation **parent**, e.g. **parent(john, jane)**. is concrete *instance* of this relation for *objects* **john** and **jane**. After declaring those facts is possible to form queries concerning relation **parent**.

```
?-parent(john, jane).
```

Example above shows, how it is possible to ask, if John is parent to Jane. Because language has this fact recorded in its environment and answer is:

```
yes
```

Similarly can be query constructed on non-existing fact:

```
?-parent(john, emily).
```

```
no
```

Answer to this query is **no** of course. Query can be also composed in a way, that we want to get some object, which is with other object in required relation.

```
?-parent(X, jane).
```

```
X = john
```

Here it is also possible to get other possible solution, so we get one more positive answer:

```
X = ann
```

After that are all possible answers exhausted, so for the next command we get:

```
no
```

Now we try to express little bit complicated query: who is mother of Jane. This query is necessary construct from two suqueries. First we limit set of solution to Jane's parents. For that purpose we use query already shown above:

```
?-parent(X, jane).
```

In variable **X** is now stored every object, who has relation **parent** to object **jane**. In our case **john** and **ann**. Now we have to limit this set of results to object, which is declared in clause **women**, so we add:

```
?-parent(X, jane), women(X).
```

As a result of these clauses we get:

```
X = ann
```

and nothing more. In framework Squander are as facts used objects, that are declared in framework's rules (see subsection 2.1.2). There are no facts declared explicitly so we do not devote to them any more.

### 2.1.2 Rules

Expressing knowledge by facts cannot be always effective. Complexity of relationships in family expressed by unconditional commands would lead to big expansion of database. Despite having unlimited memory available, searching for relevant information would have been time consuming. For this reason Prolog provides conditional expressions - *rules*.

By investigating facts is possible to derive new rule based on logical or factual context. That knowledge allows us to express facts, which are not explicitly stored in database. Let us show it in following example:

```
mother(X, Y) :- parent(X, Y), woman(X).
```

Left side of rule expressed so called **head of rule** and right side **body of rule**. This rule expresses, that **X** is mother of **Y**. Rule is only labeled generalization of last example in previous subsection 2.1.1. In next example will be shown more complicated construct:

```
brother(X, Y) :- parent(O, Y), parent(O, X), man(X).
```

Expression above means, that **X** is brother of **Y** if exists at least one object **O**, which is common in relation **parent** for both **X** and **Y** assuming **X** is a man. Mathematical interpretation of the rule is:

„For all arbitrary persons **X** and **Y**,  
 if some of the parents of **X** is **O**  
 and some of the parents of **Y** is **O**  
 and person **X** is a man,  
 then **X** is brother of **Y**.“

When calling this rule with following parameters:

```
brother(richard, Y).
```

```
Y = jane
```

Rules are main construct of framework Squander, because they defined state of object before and after computation, declares which object or object's properties can be modified etc. On the other hand these rules are not called as it is in Prolog, but they are declared as metadata for handling with objects. More about Squander's rules in chapter 4.

## 2.2 Terminology

When simplyfying in Prolog, we can say, that in every task appear *objects* and *relations*. Name of objects are called *terms* and name of relations are called *predicates*. Terms are analogous to arithmetic expressions, which point to the computed value, and predicates are analogous to name of procedures, which defines relationship between input parameters and output values, in imperative programming language.

There are two types of terms: *simple terms* consisting of constants (e.g. `ann`, `richard`, ...) and *compound terms*. Compound terms are also called *structures* is every term containing simple or compound term.

In previous subsection 2.1.2, there are predicates `parent`, `man` and `woman` as names of three relations defined by program. Predicate with name `parent` is defined as a set:  $\{(john, jane), (ann, jane), (john, richard), (ann, richard)\}$ . Next predicate with name `man` is defined as a set:  $\{john, richard\}$  and finally predicate with name `woman` is defined as a set:  $\{ann, jane\}$ .

Finally there are three types of *formulas* in Prolog:

- *atomic* - basic formulas (e.g. `parent(john, jane).`, `parent(ann, jane).`)
- *conditional command* - implication constructs  $A : -P_1, P_2, \dots, P_n$  where  $P_1, P_2, \dots, P_n$  are atomic formulas (e.g. `brother(X, Y) :- parent(0, Y), parent(0, X), man(X).`)
- *target clauses* - query type  $(? - C_1, C_2, \dots, C_n$  where  $C_1, C_2, \dots, C_n$  are targets) (e.g. `?-parent(X, jane), women(X).`)

## Chapter 3

# Meaning of annotations

Analýza a návrh implementace (včetně diskuse různých alternativ a volby implementačního prostředí).





## Chapter 4

# Framework description

Popis implementace/realizace se zaměřením na nestandardní části řešení.



## Chapter 5

# Implementation of algorithms

Popis implementace/realizace se zaměřením na nestandardní části řešení.



## Chapter 6

# Comparison

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.



## Chapter 7

# Conclusion

- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.





## Appendix A

# Testování zaplnění stránky a odsazení odstavců

**Tato příloha nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.**

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?







## Appendix B

# Pokyny a návody k formátování textu práce

**Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.**

Používat se dají všechny příkazy systému  $\text{\LaTeX}$ . Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [? ]. Tam najdete odkazy na další materiály. Většinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [? ] nebo [? ].

Existují i různé nadstavby nad systémy  $\text{\TeX}$  a  $\text{\LaTeX}$ , které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

### B.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí **figure**. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěští** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku `graphicx` příkazem `\usepackage{graphicx}`.

Budete-li zdrojový text zpracovávat pomocí programu `pdflatex`, očekávají se obrázky s příponou `*.pdf`<sup>1</sup>, použijete-li k formátování `latex`, očekávají se obrázky s příponou `*.eps`.<sup>2</sup>

Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

---

<sup>1</sup>`pdflatex` umí také formáty PNG a JPG.

<sup>2</sup>Vzájemnou konverzi mezi snad všemi typy obrázku včetně změn velikostí a dalších vymožeností vám může zajistit balík ImageMagic (<http://www.imagemagick.org/script/index.php>). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy `convert` a `identify`.

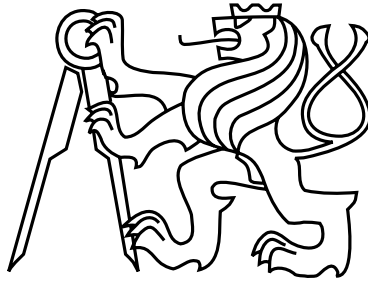


Figure B.1: Popiska obrázku

DTD	construction	elimination
	in1 A B a:sum A B in1 A B b:sum A B	case([_:A] a) ([_:B] a) ab:A case([_:A] b) ([_:B] b) ba:B
+	do_reg:A -> reg A	undo_reg:reg A -> A
*, ?	the same like   and + with empty_el:empty	the same like   and + with empty_el:empty
R(a,b)	make_R:A->B->R	a: R -> A b: R -> B

Table B.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

## B.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslicí programe Ipe [? ], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzroce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a krelit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

## B.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky B.1 vypadá takto:

```

\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
 $\mid$  & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([_:B]a)ab:A+\\
& \verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([_:B]b)ba:B+\\
\hline
 $\$$  & \verb+do_reg:A -> reg A+ & \verb+undo_reg:reg A -> A+\\
\hline
 $\$,?\$$  & the same like  $\mid$  &  $\$$  & the same like  $\mid$  &  $\$$  \\
& with \verb+empty_el:empty+ & with \verb+empty_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R -> A+\\
& & \verb+b: R -> B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab1}
\end{table}
\begin{table}

```

## B.4 Odkazy v textu

### B.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Seznam literatury je dobré zapsat do samostatného souboru a ten pak zpracovat programem bibtex (viz soubor `reference.bib`). Zdrojový soubor pro bibtex vypadá například takto:

```

@Article{Chen01,
  author   = "Yong-Sheng Chen and Yi-Ping Hung and Chiou-Shann Fuh",
  title    = "Fast Block Matching Algorithm Based on
              the Winner-Update Strategy",
  journal  = "IEEE Transactions On Image Processing",
  pages    = "1212--1222",
  volume   = 10,
  number   = 8,
  year     = 2001,
}

@Misc{latexdocweb,

```

```

author = "",
title = "{\LaTeX} --- online manuál",
note = "\verb|http://www.cstug.cz/latex/lm/frames.html|",
year = "",
}
...

```

**Pozor:** Sazba názvů odkazů je dána BibTeX stylem (`\bibliographystyle{abbrv}`). BibTeX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTeXu napovědět, která písmena nechat bez konverzí (viz `title = "{\LaTeX} --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

```

Podívejte se na \cite{Chen01},
další detaily najdete na \cite{latexdocweb}

```

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.<sup>3</sup>

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.<sup>4</sup> Zde se praví:

```

...
j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
Pozor na to, že na veškeré uvedené prameny se musíte v textu práce
odkazovat -- [1].
Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali
a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2],
statí ve sborníku [3] a html odkazu [4]:
[1] J. Žára, B. Beneš;, and P. Felkel.
    Moderní počítačová grafika. Computer Press s.r.o, Brno, 1 edition, 1998.
    (in Czech).

```

<sup>3</sup>První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znovu zavolat program `pdflatex (latex)`, který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex (latex)` lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

<sup>4</sup>Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.



- [2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User Interfaces. *Cognitive Systems*, 4(4--3):381--399, 1997.
- [3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems. In *WSCG'2000 -- Short communication papers*, pages 22--27, Pilsen, 2000. University of West Bohemia.
- [4] Knihovna grafické skupiny katedry počítačů:  
<http://www.cgg.cvut.cz/Bib/library/>

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [? ], článek v časopisu [? ], příspěvek na konferenci [? ], [www odkaz](#) [? ].

Ještě přidáme další ukázkou citací online zdrojů podle české normy. Odkaz na wiki o frameworkcích [? ] a ORM [? ]. Použití viz soubor `reference.bib`. V seznamu literatury by nyní měly být živé odkazy na zdroje. V `reference.bib` je zcela nový typ publikace. Detaily dohledal a dodal Petr Dlouhý v dubnu 2010. Podrobnosti najdete ve zdrojovém souboru tohoto textu v komentáři u příkazu `\thebibliography`.

## B.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěští se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

## B.5 Rovnice, centrováná, číselovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

Kód `$ S = \pi * r^2 $` bude vysázen takto:  $S = \pi * r^2$ .

Pokud chcete nečíselované rovnice, ale umístěné centrovane na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `$$$ S = \pi * r^2 $$$` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číselované, je třeba použít prostředí `equation`. Kód:

```
\begin{equation}
S = \pi * r^2
\end{equation}
```

```
\begin{equation}
V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \quad (\text{B.1})$$

$$V = \pi * r^3 \quad (\text{B.2})$$

## B.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí `verbatim`:

```

      (* nickname2 *)
Lego> Refine in1
      (do_reg (nickname1 h));
Refine by in1 (do_reg (nickname1 h))
    ?4 : pcddata
    ?5 : pcddata
      (* surname2 *)
Lego> Refine surname1 h;
Refine by surname1 h
    ?5 : pcddata
      (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by undo_reg (email1 h)
*** QED ***
```

## B.7 Další poznámky

### B.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“

## Appendix C

# Seznam použitých zkratek

**2D** Two-Dimensional

**ABN** Abstract Boolean Networks

**ASIC** Application-Specific Integrated Circuit

⋮



## Appendix D

### UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.



## Appendix E

# Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.





## Appendix F

# Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [? ]):



Figure F.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případně index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.