

Zpětný překlad aneb jak z binárky dostat zdroják

Petr Zemek

Fakulta informačních technologií VUT v Brně
Božetěchova 2, 612 66 Brno, ČR
<http://www.fit.vutbr.cz/~izemek>



Petr Zemek

- Ph.D. student @ VUT FIT (od 2010)
 - teorie formálních jazyků
- vývojař v AVG Technologies a člen projektu Lissom @ VUT FIT (od 2011)
 - vývoj a výzkum v oblasti zpětného překladu
- <http://petrzemek.net>



Petr Zemek

- Ph.D. student @ VUT FIT (od 2010)
 - teorie formálních jazyků
- vývojař v AVG Technologies a člen projektu Lissom @ VUT FIT (od 2011)
 - vývoj a výzkum v oblasti zpětného překladu
- <http://petrzemek.net>



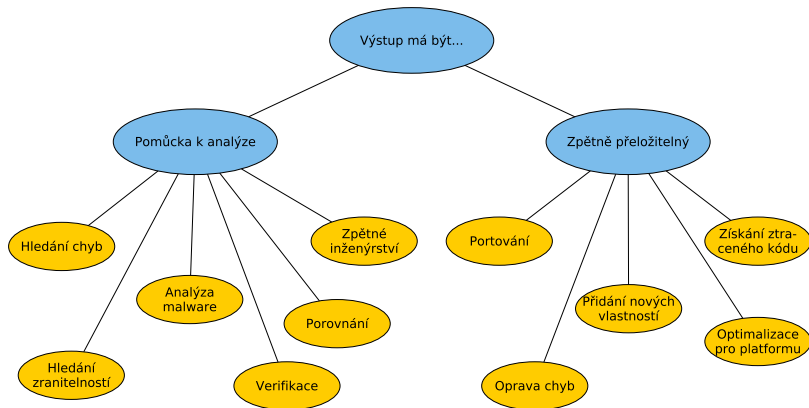
Od hamburgeru ke krávi aneb jak z binárky získat zdroják

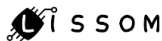
- <http://talks.petrzemek.net/barcamp2013/>

Překlad (kompilace)



Zpětný překlad (dekompilace)





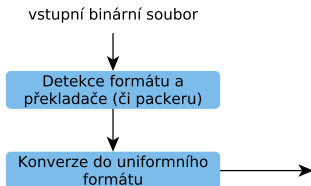
- vyvíjený od roku 2011
- Lissom @ VUT FIT ve spolupráci s AVG Technologies
- cíl: generický zpětný překlad binárního kódu
- vstup:
 - platformně závislý binární program
 - architektury: x86 (i386), ARM, ARM-Thumb, MIPS, PIC32, PowerPC
 - souborové formáty: ELF, PE
 - především programy napsané v C, ale i assembleru, C++, Delphi
 - popis platformy (CPU, ABI, knihovny, ...)
- výstup:
 - uniformní výstup v jazyku vyšší úrovně (C, Python')
 - disassemblovaný kód
 - grafické reprezentace (graf toku řízení, grafy volání)
 - statistiky

vstupní binární soubor

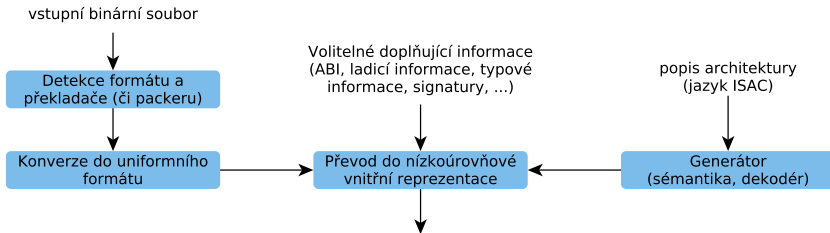


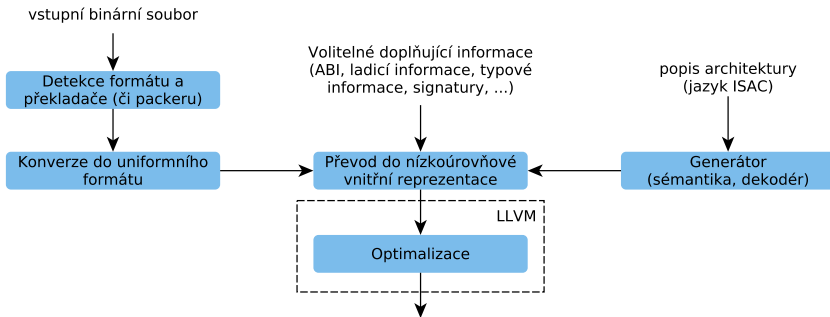
Detekce formátu a
překladače (či packeru)

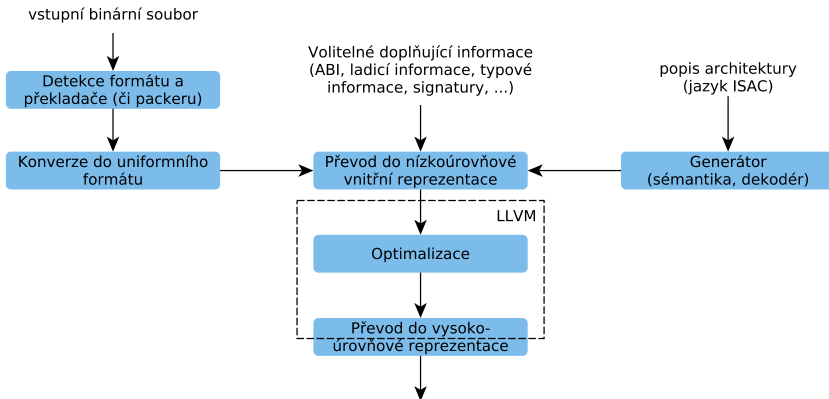


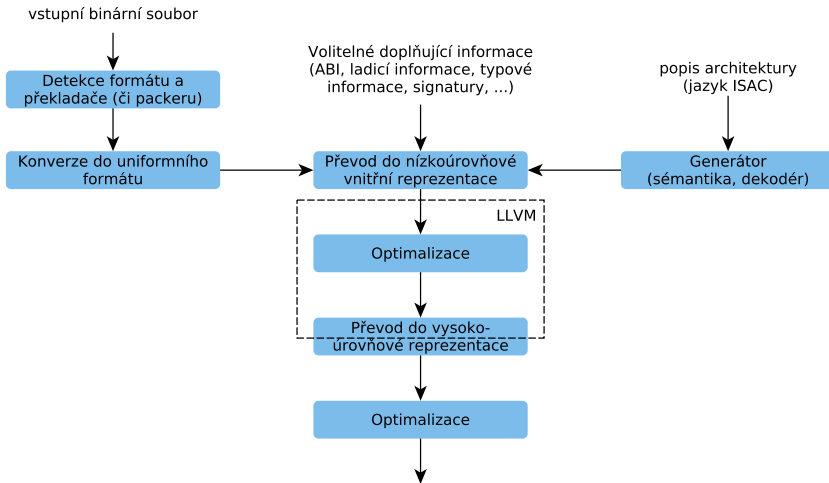


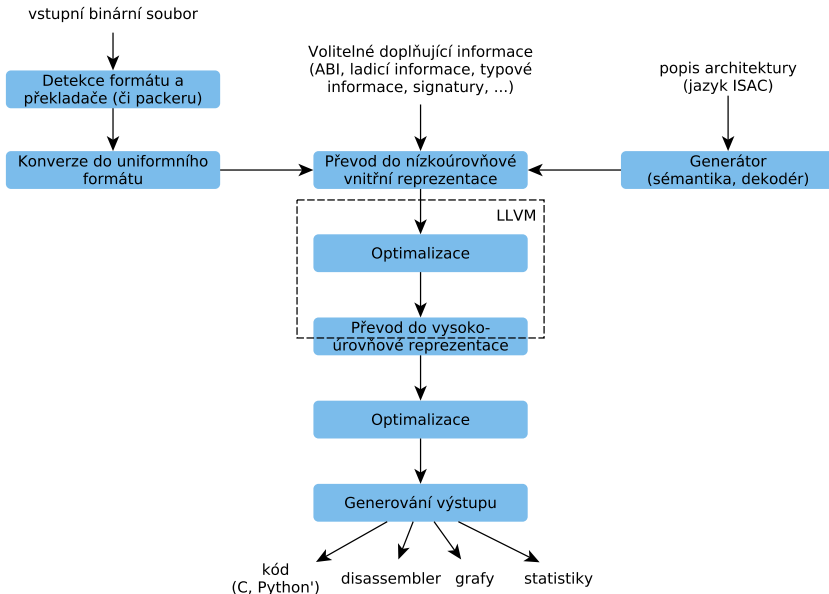
















rss


Login | Register

NEWSLETTER
Enter your email address:

Home | News | Publications | Partners | Contacts | Try Decompilation!

Welcome to the Retargetable Decompiler's Home Page

Our main goal is to create a retargetable decompiler that can be utilized for source code recovery, static malware analysis, etc. This tool is developed within the [Lissem project](#) at [Brno University of Technology](#), Czech Republic. The decompiler is supposed to be not bounded to any particular target architecture, operating system, or executable file format.

Features

- Handles all the commonly used file formats (ELF, PE).
- Currently supports the MIPS, ARM, and Intel x86 architectures.
- Can decompile to two output high-level languages: C and a Python-like language.
- Compiler and packer detection.
- Extraction and utilization of debugging information (DWARF, PDB).
- Signature-based removal of statically linked library code.
- Reconstruction of functions, high-level constructs, types, etc.
- Generation of call graphs, control-flow graphs, and various statistics.
- It is actively developed.

You can try all of these features by using our [online decompilation service](#).

Are You Interested?

If you are interested in our retargetable decompiler, please [contact us](#).

Home | News | Publications | Partners | Contacts | Try Decompilation | Legal Info

This project was supported by the research funding TACR, Alfa Programme No. TA0101.0667.
Copyright © 2013-2014 Lissem | Designed By [Free CSS Templates](#) | Validate [XHTML](#) & [CSS](#) & [RSS](#)

<http://decompiler.fit.vutbr.cz/>

*That's All
Folks!*



Původní kód:

```
#include <stdio.h>
#include <stdlib.h>

int my_sum(int i) {
    int b = rand();
    int c = rand();
    int d = b - c;
    return (b + c) * (i + d);
}

int main(int argc, char **argv) {
    int a = rand();
    int b = rand() + 2;
    int c = 0;
    if (a > b) {
        printf("TRUE");
        c = my_sum(a);
    } else {
        printf("FALSE");
        c = my_sum(b);
    }
    return a - b - c;
}
```

Dekompilovaný kód:

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int32_t function_8218(int32_t a) {
    int32_t x = rand();
    int32_t y = rand();
    return (x + a - y) * (x + y);
}

int main(int argc, char **argv) {
    int32_t apple = rand();
    int32_t banana = rand() + 2;
    if (apple > banana) {
        printf("TRUE");
        return apple - banana -
            function_8218(apple);
    }
    printf("FALSE");
    return apple - banana -
        function_8218(banana);
}
```

Původní kód:

```

1 int factorial(int n) {
2     if (n == 0)
3         return 1;
4     return n * factorial(n - 1);
5 }
6
7 int calculate(int a, int b) {
8     return a * factorial(b);
9 }
10
11 int main(int argc, char **argv) {
12     // ...
13     // ...
14     // ...
15     // ...
16     // ...
17 }
```

Dekompilovaný kód:

```

// From module: test.c
// Address range: 0x401560 - 0x401585
// Line range: 1 - 5
int32_t factorial(int32_t n) {
    int32_t result;
    if (n != 0) {
        result = factorial(n - 1) * n;
    } else {
        result = 1;
    }
    return result;
}

// From module: test.c
// Address range: 0x401587 - 0x40159c
// Line range: 7 - 9
int32_t calculate(int32_t a, int32_t b) {
    return factorial(b) * a;
}

// From module: test.c
// Address range: 0x40159e - 0x401606
// Line range: 11 - 17
int main(int argc, char **argv) {
    // ...
}
```

