VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

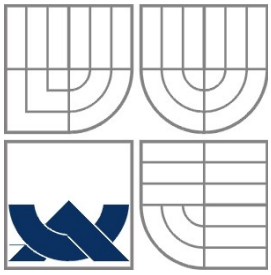
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ REKONFIGUROVATELNÉHO ZPĚTNÉHO PŘEKLADAČE

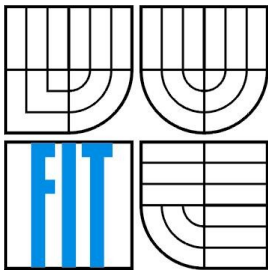
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ JÁNSKÝ
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ REKONFIGUROVATELNÉHO ZPĚTNÉHO PŘEKLADAČE

GRAPHICAL USER INTERFACE OF RETARGETABLE DECOMPILER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ JÁNSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PETER MATULA

BRNO 2014

Abstrakt

Práce se zabývá vytvářením grafického rozhraní pro zpětný překladač projektu Lissom, jenž je řízen z příkazové řádky. Jeho ovládáním překladač produkuje přeložený kód a grafu volání funkcí a kontroly toku. Zmíněné výstupy překladače zobrazuje, funkčně spojuje a jednotlivým reprezentacím výstupu přidává užitečné funkce.

Abstract

The thesis deals with creating a graphic interface for disassembler of project Lissom, which is controlled from command line. The dissasembler produces a translated code and graphs of functions calling and flow control. The mentioned outputs of the disassembler shows, functionally connects and adds to each representations of outputs useful features.

Klíčová slova

Grafické rozhraní, návrhové vzory, přenositelný kód, vykreslování grafu,zpětný překlad, Lissom.

Keywords

Graphical interface, design patterns, portable code, rendering a graph, decompilation, Lissom.

Citace

Jiří Jánský: Grafické uživatelské rozhraní rekonfigurovatelného zpětného překladače, bakalářská práce, Brno, FIT VUT v Brně, 2014

Grafické uživatelské rozhraní rekonfigurovatelného zpětného překladače

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petera Matuly.
Další informace mi poskytl pan Ing. Jakub Křoustek.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Jánský
21.5.2014

Poděkování

Děkuji celému týmu projektu Lissom za účast a rady při vývoji a návrhu grafického rozhraní. Zejména mému konzultantovi, kterým byl Ing. Jakub Křoustek.

Obsah

1 Úvod

Práce se zabývá vytvořením grafického rozhraní na již existující program zpětného překladače, který je ovládán prostřednictvím příkazové řádky.

Zpětný překladač je nástrojem pro reverzní inženýrství. V informatice je zpětné inženýrství disciplína, při které se k problému přistupuje na základě již existujících řešení. Je to náročný proces a výsledky nemusí být přehledné. Na základě tohoto problému je tedy nutné a žádoucí výsledky reprezentovat prostřednictvím grafického rozhraní. Uživateli jsou grafickým rozhraním zprostředkovány výsledky, které jsou přehledně zobrazené a užitečně funkčně propojené.

Grafické rozhraní je určeno k jednoduššímu spouštění zpětného překladače. K usnadnění zadávání různých kombinací parametrů a hodnot, které určují požadovaný výsledek zpětného překladu.

Výsledkem úspěšného běhu překladače jsou soubory zdrojových kódů, grafů a jiných důležitých informací. Výsledné soubory jsou zobrazeny pomocí složených komponent s množinou užitečných funkcí pro efektivnější práci a lepší orientaci.

Grafické rozhraní navrhováno tak, aby mohlo s překladačem fungovat na cílových platformách. To si vyžaduje specifický přístup návrhu s ohledem na podporu určitých funkcí a knihoven napříč cílovými operačními systémy. Ideálním cílem je napsat jediný vlastní kód, který by byl určen pro všechny cílové platformy. Toho lze dosáhnout za pomocí nástrojů a knihoven, které řeší problematiku přenositelnosti zdrojových kódů a kódu grafického rozhraní mezi podporovanými platformami.

Jednotlivé kapitoly této práce věnují procesu tvorby grafického rozhraní. Nejprve je potřeba znát, pro co je grafické rozhraní tvořeno a tedy jaké funkce by mělo grafické rozhraní ovládat. Tomu se věnuje kapitola s názvem „Zpětný překladač projektu Lissom“. Je zde popsán projekt Lissom a jeho zpětný překladač, zejména jeho struktura, funkce jednotlivých částí, možnosti ovládání a množina vstupů a výstupů. Poté je v práci popsána v kapitole 3 volba nástroje pro tvorbu uživatelských rozhraní na základě požadavků zpětného překladače a jeho vlastností. Následující kapitola je orientována na návrh grafického uživatelského rozhraní v rámci již zvoleného grafického toolkitu. Navržené grafické rozhraní je implementováno a popisu implementace se věnuje kapitola 4. Jsou v ní popsány konkrétní třídy, struktury a jejich mechanismy k dosažení navržených funkcí.

2 Zpětný překladač projektu Lissom

Kapitola se věnuje zpětnému překladači projektu Lissom, zejména jeho funkci,skriptům a jejich parametrům, které je nutné znát pro návrh grafického ovládání v rámci této práce. Je zde popsána funkce a princip zpětného překladače a jeho funkčních celků.

2.1 Zpětný překladač

Zpětný překladač je program, který dokáže zpětně převádět kompilovaný binární soubor (soubor strojových instrukcí závislých na platformně nebo mezikód) na kód jazyka vyšší úrovně. Kde je pak program a jeho obsah lépe čitelný.

2.2 Projekt Lissom

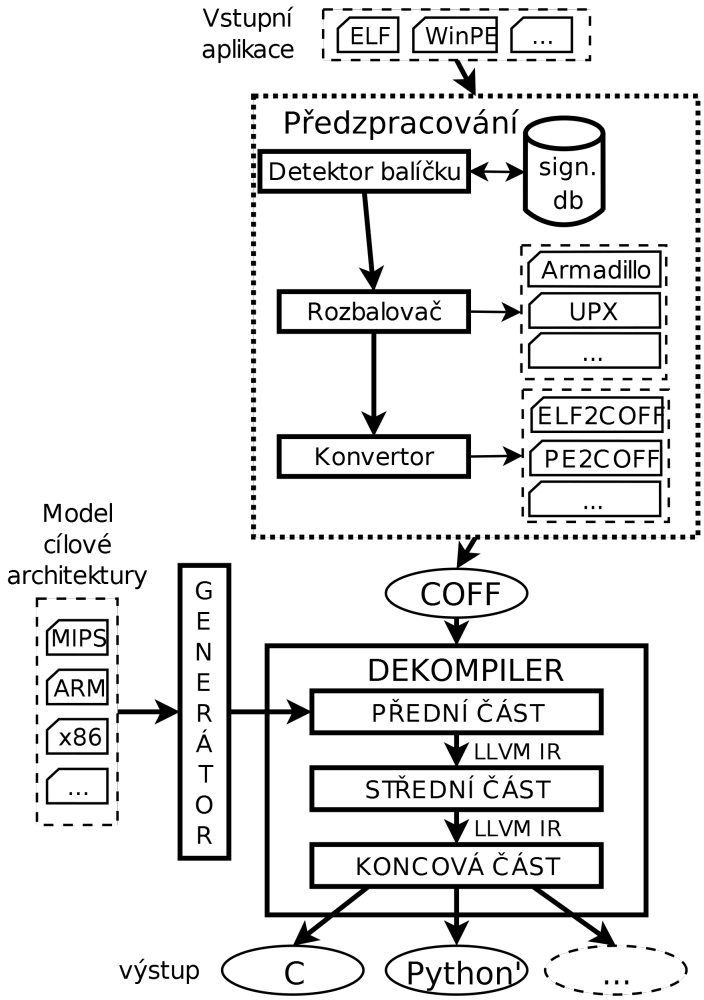
Zpětný překladač je vyvíjen v rámci projektu Lissom, který je výzkumným projektem na univerzitě Vysokého učení technického v Brně na Fakultě informačních technologii. Do projektu jsou zapojeny i soukromé firmy například firma AVG Technologies CZ, jenž se zabývá počítačovou bezpečností.

Cílem projektu je vytvoření vývojového prostředí pro návrh a simulaci procesorů spolu s obecnými platformně nezávislými prostředky pro analýzu kódu zejména škodlivého kódu na mobilních platformách. V současné době je mezi používanými procesorovými architekturami na vzestupu podíl mobilních platforem a zároveň neexistuje platformně nezávislé prostředky pro analýzu škodlivého kódu, jehož tvorba je zmíněnou situací ovlivněna.

Vývojového prostředí se skládá z návrhu hardwarové architektury a vývoje softwarového vybavení. V rámci projektu je vyvinut jazyk ADL (architecture description language) pro popis procesorů s jejich aplikačně specifickými instrukčními sadami popsané jazykem ISAC [1]. Jazyk ISAC (Instruction Set Architecture C) je pak převeden do dokumentu XML, který specifikuje architekturu procesoru. Z XML dokumentu jsou generovány nástroje i jejich knihovny. Mezi generované nástroje patří i zpětný překladač. Tím se stává zpětný překladač nezávislý na cílové platformně vstupních souborů pro zpětný překlad.

2.3 Struktura zpětného překladače

Zpětný překladač projektu Lissom funguje na bázi LLVM [2] překladového systému. V současnosti přijímá jako vstup soubory architektur Unix ELF a Windows PE, MIPS, ARM. Před zpětným překladem probíhá fáze předzpracování. Je detekován formát vstupního souboru. Soubor je rozbalen a konvertován konvertorem projektu Lissom na reprezentaci zvanou COFF (Common-Object-File-Format). COFF je vstupem do zpětného překladu. Tato fáze se skládá z 3 dílčích částí. Z přední části, střední části a koncové části. Jednotlivé části si mezi sebou předávají interní reprezentací kódu zpětně přeloženého souboru ve formátu LLVM IR (tří-adresný instrukční kód) a výsledkem zpětného překladače je zdrojový kód programu v jazyce C nebo v podobném jazyku Python [3].



Obr. 1 Koncept zpětného překladače [3]

2.4 Fáze překladu

Následující kapitola popisuje implementaci zpětného překladače projektu Lissom. Před samotným zpětným překladem je ještě vstup předzpracován. Část věnující se předzpracování má funkci konverze vstupních souborů na jednotnou reprezentaci zvanou COFF. Tento proces má oddělit rozdílné reprezentace binárních souborů a ve výsledku předat univerzální reprezentace překladači. Následkem toho je jádro překladače zjednodušeno o množství různých přístupů k rozdílným reprezentacím zkompilovaného kódů v různých podobách balíčkování.

Jádro překladače je rozděleno do tří části podle svého funkčního okruhu. Je popsána jejich funkce, způsob propojení a soubory vstupů a výstupů různých částí.

2.4.1 Předzpracování překladače

Předzpracování funguje ve třech fázích. První z nich je rozpoznání balící techniky spustitelného souboru. Detekce původně použitého nástroje pro kompresi či ochranu spustitelného souboru (tzv. packer) je na úrovni prohledávání spustitelného souboru pomocí popisů signatur těchto nástrojů. Popisy jsou buď interní (předkompilované ve vyhledávání) nebo externí (na základě externího souboru s popisovačem nebo databází popisů).

Druhá fáze je rozbalení a dekomprese balíčku. Je mnoho ochranných technik balení. V momentální situaci vzrůstá množství nových formátů balíčků a jejich způsob zabalení kódu. To má vliv na složitost a přístup této fáze.

Třetí fáze je konvertování platformně závislého formátu strojového kódů na formát pro reprezentaci univerzálního formátu platformně závislého kódu. Výsledným formátem je COFF. COFF je jako výsledek předzpracování předán jádru zpětného překladače.

2.4.2 Přední část

Přední část zpracovává soubory COFF. Je jedinou platformně závislou částí na zpětném překladači. Obsahuje dekodér instrukcí z COFF založený na cílové architektuře modelu, popsané v jazyce pro popis architektury ISAC. Přední část dekóduje COFF do sekvence LLVM IR instrukcí, které jsou nezávisle na platformě. Daná část je i zodpovědná za odstranění staticky linkovaného kódu, detekci ABI (angl. Application Binary Interface), obnovu funkce z etc a, analýzu dodatečných informací (datové sekce, ladicí informace, které zaznamenává do metadat LLVM IR). Nakonec předává výsledné LLVM IR střední části [3].

2.4.3 Střední část

LLVM IR je zde optimalizováno pomocí mnoha vestavěných optimalizací, které jsou k dispozici v LLVM a vlastními optimalizacemi (např. optimalizace smyček , neustálé množení , zjednodušení grafu toku řízení) [3].

2.4.4 Koncová část

Koncová část převede optimalizovaný meziprodukt do cílového jazyka vyšší úrovně (HLL). V současné době podporuje zpětný překladač dva cílové jazyky : C a podobný jazyku Python. Jazyk podobný jazyku Python, se liší několika rozdíly, není-li žádná podpora v Pythonu pro konkrétní stavbu , použije se konstrukce jazyka C. Převod je uskutečněn několika kroky. Vstupní LLVM IR je převedeno na jinou střední reprezentaci: zadní mezi-produktová reprezentace BIR (Back-end Intermediate Representation). Během tohoto převodu se konstruuje ovládání toku na vysoké úrovni, jako jsou smyčky, podmíněné příkazy a identifikátory . Poté získaný BIR je optimalizován, a konečně je emitován ve formě cílového HLL. Mimo jiné zadní část může vygenerovat graf volání funkcí a grafy kontroly toku pro všechny funkce [3].

2.5 Skripty a jejich parametry

Další kapitola se věnuje skriptům zpětného překladače a jejich parametrům a hodnotám parametrů.

2.5.1 Skripty

decompile.sh - Dekompiluje zadaný binární program do zvoleného cílového jazyka.
decompile-all.sh - Spustí decompile.sh s předanými parametry nad všemi spustitelnými binárkami v zadaném adresáři a podadresářích [4].

2.5.2 Parametry

Více v příloze č. 1.

-a name, --arch name	- Výběr cílové architektury [arm|arm-clang|mips|mips-clang|thumb|x86|x86-clang] (standardně: mips).
-f name, --format name	- Výběr objektového formátu [elf|pe] (standardně: elf).
-l string, --target-language string	- Cílový jazyk.
--backend-emit-cfg	- Vygeneruje control-flow graf (CFG) ze zadní části IR pro každou funkci (v .dot formátu).
--backend-emit-cg	Vygeneruje call graf (CG) ze zadní části IR pro dekompilovaný modul (v .dot formátu) [4].

2.6 Vstupy a výstupy překladače

Do překladače vstupuje zkompilovaný program v binárním souboru. S ním mohou do zpětného překladače vstoupit i další soubory s dalšími informacemi, mezi které patří soubory: ABI a ladicí informace.

Zpětnému překladači se dají předat informace o adresářích se skupinou souboru, mezi které patří: Adresář obsahující signatury a adresář obsahující typové informace.

Mezi výstupy překladu patří standardně: výsledný zpětně přeložený kód v cílovém jazyce, reprezentace binárních souboru ve formátu COFF, LLVM IR vygenerované přední částí překladače, binární reprezentace vstupního binárního souboru z přední části překladače(získáno přes llvm-as). LLVM IR, ze kterého byl vygenerován zpětně přeložený kód (optimalizované ze zadní části), binární reprezentace získaná ze zadní části překladače (získáno přes `llvm-as`) [4].

2.7 Vizualizace výstupů zpětného překladu

V aktuálním stavu je možné překladač spouštět z příkazové řádky nebo z webového rozhraní.

V prvním případě, spouštění z příkazového řádku, nejsou výstupy zobrazovány. Výstupní soubory jsou pouze uloženy.

Druhou možnosti je spouštění z webového rozhraní. Web volby zpětného překladu sice umožňuje, ale možné volby jsou velmi zjednodušené. Výstup ve webovém rozhraní je prezentován formou odkazového propojení zobrazovaného zdrojového kódu a kódu strojových instrukcí.. S grafy není umožněna další práce, jsou pouze zobrazeny. Webové rozhraní je pro větší soubory značně pomalé.

Použité rozhraní není pro reálnou práci dostatečně efektivní. Je omezeno buď množstvím úkonů s různými při analýze výsledků nebo nedává možnost dostatečně specifikovat parametry nástrojů.

3 Nástroje pro tvorbu uživatelských rozhraní

Kapitola je zaměřena na průzkum nejpoužívanějších grafických toolkitů pro tvorbu platformně nezávislého uživatelského rozhraní. Jsou zde zhodnoceny důležité vlastnosti pro vytvoření grafického rozhraní a licence, které jsou vhodné pro komerční práci. Mimo jiné jsou zde porovnávána řešení a paralelní scénáře řešení napříč srovnávanými toolkity.

Kapitola se zároveň věnuje také analýze požadavků již existujícího zpětného překladače pro který je grafické rozhraní tvořeno. Grafické rozhraní řeší jak spouštění skriptů pro zpětný překladač tak i zobrazování výstupu zpětného překladu.

3.1 Požadované prvky grafického rozhraní

Cílem práce je vytvořit grafické rozhraní k již existujícímu zpětnému překladači. V této části jsou definované oddělené požadované funkční celky pro grafické rozhraní. Zvažují se podmínky pro podporu funkcí v nástrojích pro tvorbu grafického rozhraní.

3.1.1 Spouštění zpětného překladu

Spouštění zpětného překladu je určeno množinou vstupních souborů a kombinací argumentů, popřípadě hodnotami k argumentům. To vyžaduje běžné grafické komponenty pro propojení ovladače zpětného překladu a jeho spouštění jeho skriptů. Výstup tvoří zdrojové kódy a grafy, pro které se už požaduje více než běžné elementární grafické komponenty. Pro zdrojový kód je vyžadována funkce zvýraznění syntaxe a běžné zvýrazňovací funkce jako např. párové závorky. Pro grafy je požadována metoda zobrazení grafických elementů ve vektorovém uspořádání s funkcemi přiblížení, kontextových menu speciálně pro každý element a propojení elementů s jinými grafy a kódem.

3.1.2 Editor zdrojového kódu

Pro tvorbu této komponenty musí grafický toolkit obsahovat komponentu pro zobrazení textu s možností pokročilého formátování nebo hotovou komponentu s funkcí zvýraznění syntaxí určeného HLL (High-Level Language) jazyka.

3.1.3 Vizualizace grafů

Komponenty pro modelování grafu a načítání grafu. Zde je porovnáváno více přístupů pro zobrazení grafu. Některé jsou však méně vhodné a jsou brány jako poslední možnosti. Jsou zde zmíněny již existující projekty, které využily knihoven, které jsou vhodné i pro naši práci. Projekty jsou zde uvedeny hlavně kvůli demonstraci funkčnosti řešení, ale některé z důvodů jejich vlastních licencí nemohou být pro práci použity.

Graf je možné vizualizovat více způsoby. Jednou z možností je zobrazovat výsledný soubor z zpětného překladu ve formátu SVG. Tím se pak vizualizace omezuje na zobrazení obrázku grafu bez možné interakce.

Další možností je načtení popisu grafu v souboru ve formátu DOT a následnou rekonstrukci pomocí grafických elementů pro vektorovou grafiku v grafickém toolkitu. Tato varianta je ovlivněna možností načtení a rozmístění prvků grafu knihovnou graphviz.

3.2 Přenositelnost a překladač

Cílové platformy práce jsou Linux a Windows. Přenositelnosti zdrojového kódu je dosaženo použitím jazyka C++ a implementací překladače GNU C++ v Unixu a MS Windows (součást balíčku MinGW). Výběr vhodného přenositelného grafického toolkitu řeší i přenositelnost grafického rozhraní mezi cílové platformy. Vybíralo se z množiny známějších víceplatformních grafických toolkitů, byly to Qt, wxWidgets, GTK+.

3.3 Výběr grafického toolkitu a licence knihoven

Vzhledem k nárokům na implementaci a její funkčnost je důležité vybrat správný nástroj podle více kritérií. Pro přehlednost a možnost porovnání je vytvořena tabulka č. 1 s porovnáním toolkitů, podle hledaných vlastností nebo případných dostupných knihoven. U vybraných nástrojů bylo porovnáváno několik klíčových vlastnosti v závislosti na licenci, která musí splňovat parametry pro komerční produkt.

3.3.1 Editor zdrojového kódu

Vlastnost komponenty pro zvýraznění syntaxe pro jazyky vyšší úrovně nebo schopnost a prostředky tuto komponentu jednoduše vytvořit.

Z hlediska vhodných licencí, již existujících komponent, je projekt Qt nejdražší. Editor zdrojového kódu a knihovny třetích stran jsou pro Qt knihovny pod licencí GPL nebo placené [5] na rozdíl od wxWidgets a GTK+ ve kterých je knihovna přístupná pod licencí LGL a LGPL [6] [7].

Toolkit Qt tuto skutečnost vyrovnává počtem komponent a implementovaných funkcí. Řešení vytvoření si HLL editoru v případě Qt toolkitu je splnitelné. Při zvážení nástrojů, má toolkit Qt možnost si HLL editor naprogramovat sám. S množstvím knihoven a grafických elementů v Qt je možné naprogramovat plnohodnotný HLL editor [8]. Tuto možnost nebylo třeba hledat a řešit v ostatních grafických toolkitech, jelikož hotovou komponentu pod tíženou licencí mají.

Požadovanou vlastnost HLL editoru splňují tedy všechny toolkity. I když to pro toolkit Qt znamená vlastní implementaci.

3.3.2 Vizualizace grafů

Alternativu zobrazování grafů za pomocí knihovny SVG podporují všechny toolkity. V možnosti implementovat interaktivní zobrazování grafů za pomocí knihovny Graphviz, byl nalezen příspěvek na webové stránce a funkční projekt v Qt [9]. Výsledkem porovnáváni vyplynulo, že nejlepší toolkit pro zobrazení grafů je Qt.

3.3.3 Pravděpodobné množství knihoven třetích stran

Vlastnost Pravděpodobné množství knihoven třetích stran vypovídá o pravděpodobném množství nevlastní implementace pro zajištění přenositelnosti mezi platformami. Zde je Qt asi ta nejlepší volba, pro své největší množství knihoven, z vybíraných toolkitů [10].

3.3.4 Řešení platformních odlišností a závislost na knihovnách třetích stran

V tomto ohledu je nástroj Qt vybaven velkým množstvím vlastních knihoven. Ve výsledku není třeba se spoléhat na knihovny třetích stran a jejich licenční politiku. Projekt wxWidgets je stejně zajímavým [11], ale v porovnání s Qt toolkitem není tak rozšířen. Do budoucna, kdy by se měla tato práce dále rozšiřovat, není toolkit wxWidgets tak vhodný jako Qt.

3.3.5 Volba grafického toolkitu na základě porovnávaných vlastností

Tabulka je podrobněji rozpracována v podkapitolách zabývajících se výsledkem jednotlivých průzkumů vlastnosti nástrojů. Výsledkem se stala tabulka č. 1.

Porovnávané vlastnosti	**Qt**	**wxWidgets**	**GTK+**
HLL editor - knihovna	GPL/$$$ [5]	LGL [7]	LGPL [6]
HLL editor - ručně	Ano [8]	Není třeba	Není třeba
grafy - knihovna	SVG, GPL/$$$	SVG	GPL
grafy - graphviz	Projekt qgv [9]	?	?
grafy - ručně	Ano,	SVG	Ano
Pravděpodobné množství knihoven třetích stran	Má vše své pod jednotnou vyhovující licencí.	Podobná podpora jako je u Qt.	Vše mimo grafických knihoven je řešeno buď standardními C++ knihovnami nebo knihovnami třetích stran.

Tabulka č. 1 Porovnávanými vlastnostmi napříč vyhovujícími toolkity.

A při konečném rozhodování je vybrán Qt toolkit pro jeho širokou uživatelskou základnu, množstvím knihoven, dobrou dokumentací, pro jeho inovace a rychlému vývoji, který až na menší výjimky je zpětně kompatibilní a stálý. Tím by se mělo docílit méně chyb pocházejících z knihoven třetí strany. V porovnání k ostatním hodnoceným nástrojům je Qt komplexnější ze stránky běžných funkcí a má velkou množinu svých vlastních knihoven. Ve výsledku podporuje vlastnosti HLL editor a zobrazení grafu.

4 Návrh grafického uživatelského rozhraní

Obsah této kapitoly je klasifikován jako utajený, viz licenční ujednání.

5 Implementace

Obsah této kapitoly je klasifikován jako utajený, viz licenční ujednání.

6 Testování

V kapitole testování je aplikace vyzkoušena na cílových platformách se vstupem binárních souborů, které jsou běžnými daty pro zpětný překladač v projektu Lissom.

6.1 Testovací kritéria

Testovací kritéria byla určena na základě potřeby. Komponenty a jejich prostá funkce byly specifikovány v zadání, ale během zhotovování práce byly nalezeny různé jiné potřeby a nároky, kterým bylo potřeba vyhovět.

Jednotlivá kritéria byla implementována na omezených vzorcích dat a jejich správnost funkce se může lišit ve rozdílných podmínkách. V testování je použito jako vstupu binárních souborů různé velikosti a určení na cílové platformy.

6.2 Konkrétní testovací platformy a jejich konfigurace

Program grafického rozhraní byl navržen na dvě platformy. Jejich konfigurace jsou vypsány v kapitolách Konfigurace Linux a Konfigurace Windows.

6.2.1 Konfigurace Linux

Základní informace o konfiguraci systému Linux, na kterém byla práce testována a kompilována. Pro základní popis byl využit program uname a textové soubory distribuce Linux.

```
Vydání distribuce linux:
     Fedora release 20 (Heisenbug)
Jméno jádra:
     Linux
Označení vydání jádra (Číslo) [release]:
     3.12.10-300.fc20.i686
Verze jádra (datum kompilace):
     #1 SMP Thu Feb 6 22:44:43 UTC 2014
Označení operačního systému:
     GNU/Linux
Typ počítače (hardware):
     i686
Typ procesoru:
     i686
Hardwarová platforma:
     i386
```

6.2.2 Konfigurace Windows

Základní informace o konfiguraci systému Windows, na kterém byla práce testována a kompilována. Pro základní popis byl využit program hwinfo.

```
Operační systém:
     Microsoft Windows 7 Professional Build 7601
Service Pack:
     Service Pack 1
Počítač:
     TOSHIBA Satellite U400
CPU:
     Intel Core 2 Duo P8400 (Penryn-3M POP, M0)
     2266 MHz (8.50x266.7) @ 797 MHz (6.00x132.9)
Základní deska:
     TOSHIBA Satellite U400
Chipset:
     Intel GM45 (Cantiga-GM) + ICH9M (Base)
O. Paměť:
     3072 MBytes @ 199 MHz, 6.0-6-6-18
     - 2048 MB PC6400 DDR2-SDRAM - Samsung M4 70T5663QZ3-CF7
     - 1024 MB PC6400 DDR2-SDRAM - Qimonda 64T128020EDL2.5C2
Grafická karta:
     Intel GM45/47 Chipset - Integrated Graphics 0 [B3] [Toshiba
AIS]
     Intel GMA 4500(M)(HD), 1372602 KB
     Intel GM45/47 Chipset - Integrated Graphics 1 [B3] [Toshiba
AIS]
     Mobile Intel(R) 4 Series Express Chipset Family, 1372602 KB
```

6.3 Testovací data

Testovacími daty je 10 binárních souborů. Jsou reálnými programy z praxe, takže by mělo jít spíše o pozorovanou simulaci běhu grafického rozhraní s reálnými daty, se kterými je možné potkat se v běžném provozu. Testy by měly svojí náročností objevit nedostatky, kterých jsem si nebyl při návrhu a při implementaci vědom.

10 binárních souborů je určeno pro více architektur, které zpětný překladač podporuje. Jejich velikost je v řádech stovek kilobytů.

6.4 Testované vlastnosti a jejich úspěšnost.

Testovány jsou komponenty pro zobrazování kódu a grafu a funkce, kterými jsou vybaveny. Jsou porovnány nároky na výkon, úspěšnost funkcí. Jsou zde srovnány i rozdíly účinků funkcí mezi platformami.

6.4.1 Doba načtení a zobrazení kódu a grafu

Zobrazení kódu probíhá pro soubory jazyka C a jazyka podobnému Python bez problémů. Vzhledem k jejich malé velikosti. Větší výpočetní zátěží je pro grafické rozhraní zobrazení kódů assembler, nebo llvm.

6.5 Testy

Postupně na obou platformách bude testováno 10 binárních souborů z více architektur.

6.5.1 Test č. 1

Jako vstupu byl využit binární soubor float_P60219.c_debug_all.exe, určen pro platformu MS Windows, s formátem: PE32 executable (console) Intel 80386. Velikost binárního souboru je 177,7 kB.

Platforma Linux:

Velikost zdrojového souboru C	6,6 MB
Velikost souboru assembler	5,4 MB
Velikost souboru grafu	66,5 kB
Doba trvání dekompilace	10 min. 20 sekund
Doba trvání otevření všech výstupů	3 minut a 33 sekund
Doba načtení pouze zdrojových kódů	15 sekund

Tab. č. 2 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	6,9 MB
Velikost souboru assembler	5,4 MB
Velikost souboru grafu	66,5 kB
Doba trvání dekompilace	9 min
Doba trvání otevření všech výstupů	3 min.

Tab. č. 3 Vlastnosti grafického rozhraní

6.5.2 Test č. 2

Jako vstup sloužil binární soubor x86-elf-gcc4.6.3-O0-g0—gif2tiff ve formátu ELF 32-bit LSB executable, pro architekturu Intel 80386, version 1 (SYSV), dynamicky linkovaný, pro platformu GNU/Linux 2.6.24

Platforma Linux:

Výsledná velikost zdrojového souboru C	916 kB
Velikost souboru s instrukcemi assembler	4,6 MB
Velikost Dot souboru grafu	19,4 kB
Délka trvání dekompilace	1 min. a 15 sekund
Doba trvání otevření všech výstupů	15 sekund

Tab. č. 4 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	1,1 MB
Velikost souboru assembler	4,5 MB
Velikost souboru grafu	66,5 kB
Doba trvání dekompilace	2 min. 5 sekund
Doba trvání otevření všech výstupů	53 sekund

Tab. č. 5 Vlastnosti grafického rozhraní

6.5.3 Test č. 3

Vstupem 3. testu byl soubor ms-vs-16-od-g—alloca-access.exe, ve formátu PE32 executable (console) Intel 80386, pro platformu MS Windows

Platforma Linux:

Výsledná velikost zdrojového souboru C	590 B
Velikost souboru s instrukcemi assembler	404 kB
Velikost Dot souboru grafu	19,4 kB
Délka trvání dekompilace	10 sekund
Doba trvání otevření všech výstupů	2 sekundy

Tab. č. 6 Vlastnosti grafického rozhraní

Platforma Windows:

Kompilace neproběhla úspěšně. Informace o zobrazení výsledku z Linux

Velikost zdrojového souboru C	590 B
Velikost souboru assembler	404 kB
Velikost souboru grafu	19,4 kB
Doba trvání dekompilace	9 min
Doba trvání otevření všech výstupů	3 min

Tab. č. 7 Vlastnosti grafického rozhraní

6.5.4 Test č. 4

Jako dekompilovaný soubor pro test č. 4. sloužil soubor arm-mingw32ce-gcc-O0—jpegtran, ve formátu PE32 executable (Windows CE) ARM, určen na platformu MS Windows .

Platforma Linux

Výsledná velikost zdrojového souboru C	1,3 MB
Velikost souboru s instrukcemi assembler	1,9 MB
Velikost Dot souboru grafu	20 kB
Délka trvání dekompilace	1 min. 40 sekund
Doba trvání otevření všech výstupů	10 sekund

Tab. č. 8 Vlastnosti grafického rozhraní

Platforma Windows

Velikost zdrojového souboru C	1,2 MB
Velikost souboru assembler	1,9 MB
Velikost souboru grafu	20 kB
Doba trvání dekompilace	2 min. 11 sekund
Doba trvání otevření všech výstupů	22 sekund

Tab. č. 9 Vlastnosti grafického rozhraní

6.5.5 Test č. 5

Vstupem byl soubor psp-gcc-O0-s—lame ve formátu ELF 32-bit LSB executable, určen na platformy MIPS a MIPS-II version 1 (SYSV), staticky linkovaný.

Platforma Linux:

Výsledná velikost zdrojového souboru C	2,6 MB
Velikost souboru s instrukcemi assembler	9 MB
Velikost Dot souboru grafu	27,5 kB
Délka trvání dekompilace	10 sekund
Doba trvání otevření všech výstupů	2 sekundy

Tab. č. 10 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	2,5 MB
Velikost souboru assembler	8,9 MB
Velikost souboru grafu	27 kB
Doba trvání dekompilace	8 min. 4 sekundy
Doba trvání otevření všech výstupů	2 min. 20 sekund

Tab. č. 11 Vlastnosti grafického rozhraní

6.5.6 Test č. 6

Vstupním souborem byl gnuarm-elf-gcc-O0-g—2lf ve formátu ELF 32-bit LSB executable, pro architekturu ARM, version 1, staticky linkovaný

Platforma Linux:

Výsledná velikost zdrojového souboru C	3,3 kB
Velikost souboru s instrukcemi assembler	311 kB
Velikost Dot souboru grafu	874 B
Délka trvání dekompilace	18 sekund
Doba trvání otevření všech výstupů	2 sekundy

Tab. č. 12 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	6 kB
Velikost souboru assembler	31,2 kB
Velikost souboru grafu	1 kB
Doba trvání dekompilace	30 sekund
Doba trvání otevření všech výstupů	4 sekundy

Tab. č. 13 Vlastnosti grafického rozhraní

6.5.7 Test č. 7

Vstupním souborem byl mips-android-ndk-r8-win-dijkstra_release ve formátu ELF 32-bit LSB executable, pro architektury MIPS, MIPS32 version 1 (SYSV), dynamicky linkovaný.

Platforma Linux:

Výsledná velikost zdrojového souboru C	1,1 kB
Velikost souboru s instrukcemi assembler	61 kB
Velikost Dot souboru grafu	8,4 kB
Délka trvání dekompilace	3 sekundy
Doba trvání otevření všech výstupů	1 sekundu

Tab. č. 14 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	2 kB
Velikost souboru assembler	62 kB
Velikost souboru grafu	1 kB
Doba trvání dekompilace	5 min
Doba trvání otevření všech výstupů	1 min.

Tab. č. 15 Vlastnosti grafického rozhraní

6.5.8 Test č. 8

Vstupním souborem byl arm-none-linux-gnueabi-gcc-thumb1-O0-g—cjpeg ve formátu ELF 32-bit LSB executable, na architektury ARM, EABI5 version 1 (SYSV), dynamicky linkovaný, pro platformu GNU/Linux 2.6.16.

Platforma Linux:

Výsledná velikost zdrojového souboru C	1,1 kB
Velikost souboru s instrukcemi assembler	61 kB
Velikost Dot souboru grafu	494 B
Délka trvání dekompilace	3 sekundy
Doba trvání otevření všech výstupů	1 sekundu

Tab. č. 16 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	223 kB
Velikost souboru assembler	1,7 MB
Velikost souboru grafu	9 kB
Doba trvání dekompilace	9 min. 40 sekund
Doba trvání otevření všech výstupů	18 sekund

Tab. č. 17 Vlastnosti grafického rozhraní

6.5.9 Test č. 9

Jako vstupní soubor pro test č .9 byl vybrán x86-elf-gcc4.6.3-O0-g0—tiff2ps ve formátu ELF 32-bit LSB executable, určený pro Intel 80386, version 1 (SYSV), dynamicky linkován, pro platformu GNU/Linux 2.6.24.

Platforma Linux:

Výsledná velikost zdrojového souboru C	820,9 kB
Velikost souboru s instrukcemi assembler	61 kB
Velikost Dot souboru grafu	22,3 kB
Délka trvání dekompilace	1 min. 34 sekund
Doba trvání otevření všech výstupů	17 sekund

Tab. č. 18 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	1 MB
Velikost souboru assembler	4,8 MB
Velikost souboru grafu	22 kB
Doba trvání dekompilace	1 min. 59 sekund

Doba trvání otevření všech výstupů	41 sekund

Tab. č. 19 Vlastnosti grafického rozhraní

6.5.10 Test č. 10

Jako vstup sloužil soubor ms-vs-16-od—float_basicmath_large.exe ve formátu PE32 executable (console) Intel 80386, for MS Windows

Platforma Linux:

Výsledná velikost zdrojového souboru C	104,6 kB
Velikost souboru s instrukcemi assembler	889,1 kB
Velikost Dot souboru grafu	1,3 kB
Délka trvání dekompilace	1 min. 34 sekund
Doba trvání otevření všech výstupů	17 sekund

Tab. č. 20 Vlastnosti grafického rozhraní

Platforma Windows:

Velikost zdrojového souboru C	96 kB
Velikost souboru assembler	846 kB
Velikost souboru grafu	2 kB
Doba trvání dekompilace	15 sekund
Doba trvání otevření všech výstupů	4 sekundy

Tab. č. 21 Vlastnosti grafického rozhraní

6.6 Výsledek testování

Největší zátěží bylo načtení a zobrazení grafu ze souboru dot. (tato informace byla získána při odečtení času načtení všech výstupů a času načtení pouze zdrojových kódů)

Propojení zdrojových kódů bylo také při velkém zdrojovém souboru časově náročnější. A v případě chybějících funkcí skončí na komentářích s metadaty, kde je řečeno, že jsou externími. Tedy výsledné chování je konzistentní, díky komentářům s metadaty.

Na platformě Windows byl generován pouze zdrojový kód, a to kvůli inkonzistenci knihoven Graphviz, pro grafické rozhraní a knihoven Graphviz, které používá projekt Lissom pro generování grafů v souborech dot.

Doba zvýrazňování syntaxe je na platformě Windows větší než na Linux. Možnými příčinami jsou implementace tříd pro regulární výrazy v Qt. Alternativou, jak tento problém řešit, je použití jiných knihoven pro regulární výrazy (např. Boost knihovny).

Doba otevření a zvýraznění LLVM souboru je veliká, proto nebudou zvýrazněna klíčová slova gramatiky jazyka LLVM, pro které byl běh zvýrazňování syntaxe příliš náročný.

7 Závěr

Vzniklo grafické rozhraní, které by mělo umožnit efektivnější práci s nástroji zpětného překladače projektu Lissom.

Práce splnila vytyčené body a díky frekventovaným konzultacím. Funkce grafického rozhraní byly navrženy a předvedeny se zpětnou vazbou.

Grafické rozhraní umí ovládat skript zpětného překladače pomocí předaných parametrů, pro které byly vybrány vhodné grafické komponenty.

Výsledek procesu zpětného překladu umí zobrazovat na více úrovních, které jsou mezi sebou funkčně spojeny. Tím je reprezentován výstup grafu jako celistvý výsledek, který je tak lépe pochopitelný.

Výsledek práce by mohl být dále rozvíjen, a to díky použití nástrojů majících velikou oblibu mezi vývojáři, návrhových vzorů, jež by měly poskytnou kódu přehlednost a rychlejší další implementace. Projekt je ve své implementaci rezervován pro další rozvoj. Ve své konečné podobě je několik bodů, které by se dali implementovat v budoucnu lépe.

Budoucí rozvoj by určitě obnášel implementaci některých funkcí v neblokujících procesech nebo vláknech, tak aby bylo grafické rozhraní uživateli přívětivější.

Analýza kódu pro zvýraznění by mohla být implementována pomocí konečného automatu.

Modul vykreslení grafu by mohl být více přehlednější pomocí interaktivního zvýrazňování a vykreslování grafu po menších částech.

Zvýraznění kódů by mohlo mít nastavení barev v externích souborech pro uživatelské nastavení a jeho přenositelnost mezi více instalacemi.

Literatura

[1] Lissom [online]. 2014 [cit. 2014-01-29]. Dostupné z: http://www.fit.vutbr.cz/research/groups/Lissom/

[2] *The LLVM Compiler Infrastructure* [online]. 2014 [cit. 2014-01-29]. Dostupné z: http://llvm.org/

[3] Ďurfina, L., Křoustek, J., Zemek, P.: Psyb0t Malware: A Step-by-Step Decompilation Case Study, In: 20th Working Conference on Reverse Engineering (WCRE), Koblenz, DE, IEEE CS, 2013, s. 449-456, ISBN 978-1-4799-2930-6

[4] Jak zprovoznit a používat dekompilátor. Interní wiki projektu Lissom - Decompiler [online]. 2013 [cit. 2014-01-26]. Dostupné z: http://212.4.138.237/redmine/projects/decompiler/wiki/DecompilerHOWTO

[5] *QScintilla: a Port to Qt v4 and Qt v5 of Scintilla* [online]. 2013 [cit. 2014-01-29]. Dostupné z: http://pyqt.sourceforge.net/Docs/QScintilla2/index.html

[6] *GtkScintilla* [online]. 2014 [cit. 2014-01-30]. Dostupné z: http://codebrainz.github.io/GtkScintilla/

[7] *WxStyledTextCtrl Class Reference: Scintilla Text Editor* [online]. 2014 [cit. 2014-01-30]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_styled_text_ctrl.html

[8] Syntax Highlighter Example. *Qt Project* [online]. 2013 [cit. 2014-01-30]. Dostupné z: http://qt-project.org/doc/qt-5.0/qtwidgets/richtext-syntaxhighlighter.html

[9] Qgv: Qt GraphViz display. Google code [online]. [cit. 2014-01-29]. Dostupné z: http://code.google.com/p/qgv/

[10] *Qt Project: Documentation* [online]. 2013 [cit. 2014-01-26]. Dostupné z: https://qt-project.org/doc/qt-5.1/qtdoc/index.html

[11] *WxWidgets: Documentation* [online]. 2013 [cit. 2014-01-30]. Dostupné z: http://www.wxwidgets.org/docs/

[12] How-to: Use Graphviz to Draw Graphs in a Qt Graphics Scene. D. LAZARO, Steve. Http://www.mupuf.org/ [online]. [cit. 2014-01-26]. Dostupné z: http://www.mupuf.org/blog/2010/07/08/how_to_use_graphviz_to_draw_graphs_in_a_qt_graphics_scene/

[13] Qt project: QDockWidget Class Reference. *Qt Project* [online]. 2013 [cit. 2014-01-26]. Dostupné z: http://qt-project.org/doc/qt-4.8/qdockwidget.html

[14] C. NORTH, Stephen. GRAPHVIZ OPEN SOURCE TEAM. Cgraph Tutorial. AT&T Shannon Laboratory, Florham Park, NJ, USA, 2013. Dostupné z: http://www.graphviz.org/pdf/Agraph.pdf

[15] R. GANSNER, Emden. GRAPHVIZ OPEN SOURCE TEAM. Graphviz Library Manual. unknow, 2013. Dostupné z: http://www.graphviz.org/doc/libguide/libguide.pdf

[16] Vzory: Návrhové vzory (design paterns). DVOŘÁK, M. *Objektová analýza, návrh a programování* [online]. 2005 [cit. 2014-05-19]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory

Příloha č. 1

Parametry skriptů:

-d, --debug	- Výpis ladicích informací.
-c, --cleanup	- Odstraní dočasné soubory produkované scriptem.
-o file, --output file	- **Výstupní soubor.**
-g dir, --signatures dir	- Adresář obsahující signatury.
-t dir, --types dir	- Adresář obsahující typové informace.
-A file, --abi file	- Soubor s ABI.
--debug-info	- Přeloží zdrojový soubor s ladicími informacemi (-g).
--strip	- Odstraní z přeloženého programu symboly.
--dot	- Vygeneruje .dot soubor v decfrontu.
--compiler-opts level	- Nastavení optimalizační úrovně u překladače jazyka C (standardně: -O2).
--keep-unreachable-funcs	- Ponechá ve výstupu funkce, které nejsou dosažitelné z hlavní funkce (typicky main).
--backend-no-debug-comments	- Vypne generování ladicích komentářů ve výstupu z back-endu.
--backend-semantics string	- V back-endu se použijí sémantiky ze zadaného seznamu. Jednotlivé sémantiky jsou odděleny čárkou.
--backend-enabled-opts string	- Spustí pouze optimalizace ze zadaného seznamu. Jednotlivé optimalizace jsou odděleny čárkou.
--backend-disabled-opts string	- Vypne optimalizace ze zadaného seznamu. Jednotlivé optimalizace jsou odděleny čárkou.
--backend-no-opts	- Vypne optimalizace v backendu.
--backend-aggressive-opts	- Zapne tzv. agresivní optimalizace. Ty mohou způsobit, že kód bude čitelnější, ale nebude korektní.
--backend-var-renamer string	- Použitý přejmenovávač proměnných [address|hungarian|readable|simple|unified] (standardně: readable)."
--backend-no-var-renaming	- Vypne přejmenování proměnných v backendu.
--backend-no-symbolic-names	- Vypne konverzi čísel na symbolická jména.
--backend-keep-all-brackets	- Ve výstupu zůstanou všechny závorky, i když nejsou potřeba.
--backend-keep-library-funcs	- Ponechá ve výstupu funkce ze standardních knihoven.
--backend-no-time-varying-info	- Ve výstupu se neobjeví informace, které jsou závislé na čase (např. data).
--backend-cfg-prefix string	- Prefix každého souboru obsahujícího CFG (standardně: cfg.).
--backend-cg-file-name string	- Jméno výstupního souboru, do kterého je CG vygenerován (standardně: call-graph.dot).
--backend-unify-labels	- Přejmenuje návěští příkazů a jména uzlů v CFG/CG grafu (nutné použít při testech).
--backend-force-module-name string	- Použije zadaný název modulu místo toho, který vygenerovala přední část.
--backend-call-info-obtainer string	- Jméno použitého poskytovatele informací o voláních funkcí (standardně: optim).
--backend-arithm-expr-evaluator string	- Jméno použitého vyhodnocovače aritmetických výrazů (standardně: C). [4]