Posudek diplomové práce

Matematicko-fyzikální fakulta Univerzity Karlovy

Autor práce	Jan Dubský		
Název práce	Extensible disassembler with support for interactive instruction reordering		
Rok odevzdání	2022		
Studijní program	Informatika	**Studijní obor**	Softwarové systémy

Autor posudku	Lubomír Bulej	**Role**	vedoucí
Pracoviště	KDSS MFF UK		

Text posudku:

Disassembler je jedním z hlavních nástrojů používaných při reverzním inženýrství programů (např. v oblasti počítačové bezpečnosti). Disassembler převádí strojový kód programu v binární (typicky spustitelné) podobě do textové podoby čitelné člověkem. Pochopit chování programu ze strojového kódu je však velmi náročná úloha—především kvůli ztrátě abstrakcí běžných ve vyšších programovacích jazycích.

Jedním z faktorů, které znesnadňují pochopení chování strojového kódu generovaného překladačem je fakt, že překladač může uspořádat strojový kód libovolným způsobem—pokud zachová sémantiku programu. Praktický důsledek je, že ve výsledném kódu jsou blízko sebe (nebo za sebou) nezávislé instrukce např. z různých bloků kódu ve funkci, přestože v původním programu jsou u sebe např. příkazy, které pracují s jednou hodnotou. Pro lepší srozumitelnost by bylo vhodné, kdyby uživatel mohl přeuspořádat instrukce tak, aby lépe odrážely programátorský pohled na věc, což však současné disassemblery neumožňují.

V tomto kontextu bylo cílem práce navrhnout experimentální disassembler, který by uživateli umožnil lokálně přerovnat instrukce programu a učinit tak kód srozumitelnější, ale neumožnil (a nebo alespoň nějak indikoval) přesuny, které by změnily sémantiku programu. Zároveň bylo cílem umožnit uživateli běh programu v omezeném rozsahu simulovat a v principu podporovat různé architektury procesorů (a nevázat tedy vnitřní návrh disassembleru na jeden konkrétní typ procesoru).

Ač to nemusí být první pohled zřejmé, jedná se o velmi netriviální zadání. Pokud bychom se spokojili s pouhým přerovnáním řádků v textovém výstupu disassembleru, nebylo by potřeba "rozumět" tomu, co instrukce dělají. Pokud však chceme umožnit pouze platná přeuspořádání, při kterých nedojde např. ke kolizím registrů, je potřeba zcela jiný přístup. Stejně tak samotná podpora různých procesorů není velký problém, pokud bychom vyžadovali pouze vygenerování textového přepisu strojového kódu. Pokud má však disassembler podporovat také přerovnání instrukcí pro různé procesory a být schopen instrukce odsimulovat, jedná se o výrazně složitější problém.

Vedle textu práce, který podrobně rozebírá řešené problémy, je výsledkem také prototypová implementace disassembleru, který umožňuje přerovnávat instrukce v rámci základních bloků a umožňuje simulovat běh programu. V současné podobě disassembler podporuje procesory RISC V, přičemž vnitřní rozhraní jsou navržena tak, aby bylo možné disassembler rozšířit o podporu jiných procesorů. Netriviální rozsah a charakter práce odráží i prototypová implementace v rozsahu přes 13 kLOC v jazyce Go, přestože disassembler (cíleně) poskytuje pouze nejzákladnější funkce nutné pro demonstraci celého přístupu.

Text práce je v angličtině a je rozčleněn do kapitol, které vedle počáteční analýzy pokrývají návrh platformově nezávislé reprezentace instrukcí, parsování strojového kódu, optimalizaci vnitřní reprezentace instrukcí, přeuspořádání instrukcí a nakonec emulaci instrukcí. Text trochu trpí množstvím překlepů či nekonzistentním používáním 1. os. j.č. a 3. os. m.č., ale čitelnosti textu to zásadním způsobem neškodí a jako větší slabinu vnímám spíše fakt, že autor některé aspekty rozebírá velmi ze široka i situacích, které by znalý čtenář nepotřeboval vysvětlovat (např. bitové a logické operace procesoru a řada dalších), a které způsobují, že pro stromy občas není vidět les. Organizace textu neumožňuje taková místa jednoduše vynechat, protože jsou často proložena

úvahami, které souvisí s problémem. Problém s přehlcením informacemi naštěstí výrazně zmírňují shrnutí na konci každé kapitoly, která velmi oceňuji.

Po technické stránce nemám k práci vážné výhrady. Aspekty řešení, které se netýkají hlavní podstaty práce (tj. vnitřní reprezentace a přerovnávání instrukcí) autor řeší zavedenými postupy, tj. kód je po naparsování rozložen na základní bloky v rámci nichž pak probíhá analýza závislostí mezi instrukcemi. Jako elegantní vnímám reprezentaci instrukcí pomocí efektů, které umožňují sledovat závislosti mezi instrukcemi (a instrukce simulovat) až na úrovni vnitřní reprezentace, nikoliv v rámci podpory pro konkrétní procesor. Potenciální slabinou je nemožnost přesouvat instrukce mezi základními bloky (neměl by být problém přesunout instrukci do dominujícího bloku, pokud nedojde ke kolizi mezi registry) a rovněž podpora pro kód s delay sloty, které jsou běžné u starších RISCových procesorů, přestože z návrhu architektury moderního procesoru RISC V vyplývá, že delay sloty jsou spíše záležitost mikroarchitektury a nepatří do moderní ISA.

Za určitou slabinu považuji absenci bližšího srovnání autorem navržené vnitřní reprezentace se spustitelnou vnitřní reprezentací ESIL (Radare2) nebo REIL (Angr). Rovněž by bylo zajímavé zamyslet se nad možnostmi využití knihovny Capstone (univerzální disassembler) či Unicorn (univerzální emulátor). Na druhou stranu musím podotknout, že práce rozhodně netrpí tím, že by byla malá rozsahem a záběrem, a že jejím primárním cílem bylo vyzkoušet něco, co jiné nástroje nedělají. Proto to nepovažuji za vážný problém.

Práce jako celek úspěšně splnila zadání a autor bezpochyby prokázal svou schopnost samostatně řešit netriviální problém, stejně jako schopnost toto řešení zpracovat na vysoké technické úrovni.

Práci doporučuji k obhajobě.

Práci nenavrhuji na zvláštní ocenění.

V Praze dne 30. 8. 2022

Podpis: