VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

http://fit.vut.cz

Projektová praxe 2019/2020

Porovnání a experimentální vyhodnocení různých přístupů ke statické analýze programů

Tomáš Beránek*

Abstrakt

Cílem této práce je experimentální vyhodnocení různých přístupů ke statické analýze programů, konkrétně se jedná o způsoby nasazení statického analyzačního nástroje Facebook Infer na reálný software. Ověření vlastností a vhodnosti nástroje probíhá spouštěním analýz na různých typech softwaru. Hodnotí se schopnost software zanalyzovat, škálovatelnost, poměr skutečných chyb ku potenciálním chybám a úplnost nalezení chyb. Vhodná kombinace těchto vlastností je pro nasazení v praxi klíčová. Špatná škálovatelnost a nízký poměr skutečných chyb ku potenciálním (respektive množství problémů ke kontrole), lze do jisté míry kompenzovat integrováním analýzy již do vývoje softwaru, proto se tato práce zaměřuje také na integraci nástroje Infer s verzovacím systémem git a automatickým spouštěním nad repozitáři. Výhodou nástroje Infer je vysoká škálovatelnost pro základní analyzátory (Infer poskytuje i analyzátory ve vývoji, označeny jako experimentální) a proto lze analyzovat i poměrně objemné kódy. Poměr počtu skutečných chyb ku potenciálním je však v jistých případech velice nízký a tudíž při použití Inferu pro analýzu celého softwaru najednou je počet hlášení k ověření velkým problémem. Nasazením nástroje Infer na software, ať už na hotový nebo při vývoji, se zvýší kvalita produkovaného softwaru. Při použití automatizované analýzy při vývoji, jsou navíc zdroje potřebné pro její zavedení a běh téměř zanedbatelné.

Klíčová slova: statická analýza — Facebook Infer — automatizace statické analýzy

Přiložené materiály: https://github.com/TomasBeranek/IP1_supplementary

*xberan46@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Cílem této projektové praxe je experimentální vyhodnocení různých přístupu ke statické analýze programů. Se zvyšující se složitostí softwaru, je stále obtížnější produkovat bezchybný kód. Z důvodu velkého počtu možných cest v kódu je velice obtížné otestovat software dynamickými testy. Zde přichází na řadu formální verifikace, která dokáže prokázat bezchybnost programu (ve smyslu absence běhových chyb) bez jeho spuštění, pouze zkoumáním zdrojových souborů (statická analýza). Ačkoliv je taková myšlenka užitečná, její převedení do praxe je velice obtížné. Nástroje jsou vytvářeny jako kompromis mezi rychlostí, jednoduchým použitím a schopností nalézt veškeré chyby. Existují však i nástroje, které jsou schopny dokázat bezchybnost programu, ale jejich použití je většinou pro reálné softwary nepoužitelné - mohou být velmi výpočetně náročné nebo je nutné je pracně přizpůsobit na konkrétní software. I tak ale najdou v jistých odvětvích své použití, zejména v letectví, medicíně, kosmickém průmyslu atd. Existuje spousta nástrojů pracujících na principu statické analýzy softwaru, ale ne každý lze použít na všechny typy softwaru, proto se tato projektová praxe zaměřuje především na zhodnocení statického analyzačního nástroje Facebook Infer, jeho použití na reálný software a integraci do vývoje. Hodnotí se, zda je Infer možné spustit na softwaru, schopnost škálovat, poměr opravdových chyb ku falešným hlášením, schopnost nalézt všechny chyby a případně jeho vhodnost na integraci do vývojového procesu. Vhodná kombinace těchto vlastností je pro nasazení v praxi klíčová. Samotné testování nástroje Infer probíhalo analyzováním reálného softwaru i uměle vytvořených situací k ověření určitých vlastností.

1.1 Existující přístupy ke statické analýze

Nástroj Facebook Infer je v současné době nasazen ve firmě Facebook při vývoji mobilních aplikací více v kapitole Nasazení Inferu ve Facebooku. V praxi se však využívají i jiné statické analyzační nástroje, například firma Red Hat využívá celé sady statických analyzačních nástrojů jak pro průběžné, tak pro celkové testování softwaru [1]. Firma Honeywell využívá obdobného principu a navíc se snaží přizpůsobit statické analyzační nástroje na kontrolu assert maker v jejich softwaru více v kapitole Přizpůsobení Inferu na kontrolu assertů.

2. Nástroj Facebook Infer

Facebook Infer [2] je nástroj pracující na principu statické analýzy programů (zkoumá pouze zdrojové kódy, bez jejich spuštění). Infer dokáže identifikovat různé typy problémů - úniky paměti, dereference NULL ukazatelů, nepovolené přístupy do paměti, dosažitelnost anotací, mrtvý kód, invariance kódu ve smyčkách, data race, hladovění atd. Pomocí rozšíření, která byla vyvinuta na VUT FIT, lze detekovat například deadlock - rozšíření L2D2 [3] (A Low Level Deadlock Detector) nebo porušení atomicity sekvence příkazů - rozšíření Atomer [4] (Atomicity Violations Analyser). Infer je složen z řady nezávislých analýz, které lze podle potřeby zapnout/vypnout. Každá analýza se zaměřuje na hledaní určitého typu chyb a také podporuje pouze určité programovací jazyky. Většina analýz však podporuje jazyky Java, C, C++ a Objective-C (vždy je podporován minimálně jeden z těchto jazyků a jiné podporovány nejsou).

Nástroj Infer není úplný, co se týče nalezení chyb (angl. "sound"). To znamená, že pomocí Inferu nelze dokázat, že se v analyzovaném softwaru nevyskytuje chyba[1]. To je zapříčiněno tím, že Infer je vytvořen tak, aby dobře škáloval a bylo jednoduché jej použít. Nástroje, které naopak jsou schopny dokázat absenci chyby ve zdrojovém kódu (angl. Sound Static Analyzers) jsou pro většinu reálných softwarů nepoužitelné, jelikož jsou velmi výpočetně náročné nebo je nutné je ručně přizpůsobit na konkrétní software. Avšak v jistých oblastech, kde je absence chyb klíčová (kybernetická bezpečnost, lékařství, letectví, kosmický průmysl a jiné), najdou tyto nástroje využití např. použití Frama-C na X.509 parser [5]. Analýza nástrojem Infer probíhá ve dvou fázích. V první fázi se zachytí překladové příkazy buď přímo nebo ze sestavovacího systému (make, cmake atd.). A v druhé fázi probíhá samotná analýza pomocí jednotlivých analyzátorů.

[1] zde se chybou myslí např. pád programu, nikoliv chybná implementace, ve smyslu, že program nedělá to co má

Překladové příkazy jsou potřeba pro převod zdrojových souborů do vlastního mezikódu, používaného Inferem. Tento převod se velmi podobá překladu a proto se využívá vnitřního překladače (clang pro jazyky z rodiny C a javac pro Javu) pro získání informací o souborech, jako je například CFG (Control Flow Graph) pro každou metodu/funkci, CG (Call Graph) atd. Infer podporuje celou řadu překladových systémů [6]. Překladové příkazy jsou provedeny jako obvykle a navíc použity pro vytvoření mezikódu. Jelikož Infer využívá vždy svůj interní překladač, mohou nastat problémy s kompatibilitou viz. kapitola Nasazení Inferu na reálný software. Pokud není žádný soubor překládán, nejsou vytvořeny interní soubory v mezikódu potřebné pro analýzu. Zde je třeba si uvědomit, že většina překladových systémů nepřekládá již přeložené soubory, které jsou aktuální. V takovém případě nepřekládané soubory nebudou Inferem analyzovány. Proto, pokud si chceme být jisti, že Infer je spuštěn na celém projektu, je potřeba jej nejdříve vyčistit (make clean, mvn clean atd.). Infer lze spustit, aby přeložil a zanalyzoval pouze nově udělané změny, na tomto principu funguje integrace s Gitem a automatické spouštění, podrobnější informace jsou v kapitole Integrace nástroje Infer s verzovacím systémem Git. Veškeré soubory vytvořené Inferem jsou uloženy ve složce infer-out/ v adresáři, ve kterém byl spuštěn (to lze samozřejmě vhodnými přepínači změnit).

Ve druhé fázi Infer analyzuje soubory v mezikódu vytvořené během první fáze. Pokud je již vytvořena složka se soubory v mezikódu, pak je na nich možné spustit libovolné analyzátory opakovaně dle potřeby. Analýza probíhá odspodu, to znamená, že nejdříve jsou analyzovány nezávisle jednotlivé funkce, které jsou poté skládány v závislosti na volání. Díky tomuto je možné analyzovat i nedokončené projekty nebo jednotlivé soubory nezávisle. Avšak překlad by měl být vždy úspěšný. Infer sice i při selhání překladu dokáže vytvořit soubory v mezikódu, které odpovídají funkcím přeloženým před selháním, ale při spuštění analýzy nevíme, které funkce byly skutečně analyzovány a které ne. Tato skutečnost se může hodit při analýze projektů, jejichž překlad není kompatibilní s vnitřním překladem Inferu. Lepší variantou však je nahradit nepřeložitelné konstrukce takovými přeložitelnými, u kterých bude hodnota, kterou mají vracet, neznámá. Infer se poté k návratovým hodnotám těchto konstrukcí chová, jakoby mohli nabývat libovolné hodnoty v daném rozsahu a na základě toho provádí analýzu všech možných případů. Díky této nadaproximaci většinou nenastane případ, kdyby kvůli neimplementované/nepřeložitelné konstrukci byla vynechána

nějaká chyba. Chyb se však může objevit více. Z toho vyplývá, že množina chyb zjištěných při analýze jednotlivých funkcí je nadmnožinou chyb při analýze celého projektu (čím větší část projektu se analyzuje, tím více falešných chyb se vyloučí). Vysoká rychlost analýzy je zajištěna paralelním zpracováním jak jednotlivých analýz, tak souborů v rámci dané analýzy. Možnost paralelního zpracování souborů funguje díky analýze zespoda nahoru, kterou Infer využívá.

2.1 Nasazení Inferu ve Facebooku

Nástroj Infer je v současné době využíván ve Facebooku [7] hlavně při vývoji mobilních aplikací jako jsou Messenger, Instagram, Facebook aplikace, jak na platformu Android tak iOS. Jelikož jsou webové aplikace napsány převážně v PHP nebo Javascriptu, není možné na ně Infer zatím použít, proto je Infer nasazen na mobilní aplikace psané v Javě, C++ nebo Objective-C. U mobilních aplikací se také klade větší důraz na "bezchybný" kód, protože u webových aplikací je oprava chyb mnohem jednodušší. Opravené verze webových aplikací lze nahrát přímo na servery, se kterými komunikují přes prohlížeče zákazníci a tak distribuce opravené verze je jednoduchá. U mobilních aplikací lze novou verzi pouze nahrát do Google Play nebo App Store a už je čistě na uživatelích, zda aktualizaci povolí nebo ne.

Kvůli neustále se měnícím požadavkům, ze strany uživatelů, Facebook využívá kontinuální vývoj a tomu musí být také přizpůsobeno nasazení Inferu. Analýza probíhá jak průběžně na změnách, tak jednou za čas na celém softwaru. Analýza Inferem je zcela automatizovaná, aby se neplýtvalo časem vývojářů. Každá změna vytvořená vývojáři musí projít regresivními testy, poté je zpřístupněna ostatním vývojářům k diskuzi (ohledně chyb, čitelnosti, robustnosti atd.). V této fázi je také změna zanalyzována Inferem a do několika minut jsou výsledky zpřístupněny jako komentáře k částem kódu, kde se může vyskytovat chyba. Komentáře jsou vloženy, jako kdyby je vkládal člověk, který kód kontroloval. Součástí komentářů je také zpětné sledování chyby pro rychlejší opravu nebo její vyloučení. Během analýzy změn se analyzují pouze změněné funkce a funkce na nich závislé (u klasické verze Inferu tohoto nelze docílit). Přes noc probíhají celkové testy se všemi změnami, které se uskutečnily přes den. Díky průběžným testům se zmenšuje počet případů, kdy je programátor nucen se vracet ke staršímu kódu, aby opravil chyby.

2.2 Dílčí analyzátory Inferu

Nástroj Infer poskytuje řadu na sobě nezávislých analyzátorů. Ty se rozdělují na základní (spustí se vždy) a experimentální (ty je vždy nutné zapnout ručně). Základní analyzátory jsou annotation-reachability, bi-abduction, fragment-retains-view, immutable-cast, liveness, ownership, printf-args, racerd, siof a uninit. Experimentální analyzátory jsou inferbo, eradicate, quandary, starvation a litho. Níže jsou detailněji popsány analyzátory, zabývající se nejběžnějšími chybami:

bi-abduction analyzátor je založený na bi-abdukci nad separační logikou [8]. Slouží k detekci chyb s ukazateli, jako například přístup přes ukazatel, označen jako NULL, úniky paměti a chybějící testování na NULL ukazatel po alokaci paměti. Podporované jazyky jsou C,C++,Obj-C a Java. Nevýhodou tohoto analyzátoru je skutečnost, že při první nalezené chybě v aktuální analyzované funkci přeruší analýzu. Takto zůstane část od prvního hlášení do konce aktuální funkce nezanalyzovaná. V takovém případě je nutné chybu vyřešit a zopakovat analýzu. Když chybu nelze odstranit (např. nejedná se o opravdovou chybu nebo by její odstranění bylo náročné) nastává problém. Řešením tohoto problému se zabývá kapitola Odstranění zastavení analýzy u analyzátoru bi-abduction.

pulse je experimentální analyzátor kontrolující běhové chyby jako např. přístup k již uvolněné paměti nebo opětovné uvolnění paměti. Podporovaný je pouze jazyk C++. Ve verzi Inferu 0.17.0 je do analyzátoru pulse zahrnut analyzátor ownership.

racerd analyzátor hledá synchronizační chyby typu data race. Podporovaný jazyk je pouze C++ a Obj-C. Aby byl analyzátor aktivován, musí být metody (které je potřeba zanalyzovat) označeny jako synchronized.

liveness analyzátor detekuje nepoužité proměnné a funkce (angl. Dead Store). Podporuje jazyky C, C++ a Obj-C.

inferbo je experimentální analyzátor založený na abstraktní interpretaci. Slouží k detekci přístupu za hranice pole. Analyzátor hůře škáluje, což je způsobeno poměrně přesným sledováním intervalů možných hodnot proměnných, návratových hodnot atd. Proto byl využit pro kontrolu assertů, více v kapitole Přizpůsobení Inferu na kontrolu assertů. Podporovaný jazyk je pouze C, ale je plánováno přizpůsobení i na C++ a Java.

3. Nasazení Inferu na reálný software

Existují dva základní přístupy, jakými lze využívat Infer. Prvním je jednorázová analýza již hotového (nebo alespoň z části hotového) softwaru. A druhým je nasazení při vývoji (od začátku). Každý z přístupů se potýká s jinými problémy. Tato projektová praxe se nejdříve zaměřovala na analýzu již hotového softwaru a ke konci i na nasazení Inferu do vývoje.

Při použití Inferu jednorázově na hotový software nastává řada problémů, které je třeba vyřešit. Prvním problémem bývá zpřístupnění překladových příkazů, které Infer potřebuje k analýze. V praxi překladový proces může být velice složitý a používat různorodé překladové nástroje. Muže být také sloučen se sestavujícím procesem (v takovém případě je nutné extrahovat překladové příkazy). Nástroj Infer bohužel podporuje jen několik málo základních sestavovacích nástrojů a získaní překladových příkazů bývá často velice náročné nebo i nemožné. Dalším problémem může být nekompatibilita vnitřního překladače Inferu s použitým překladačem. Infer interně používá clang 8.0.0 pro C/C++/Objective-C a javac 8 pro Javu (toto platí pro Infer v0.17.0, pro testování v projektové praxi byla použita verze 0.16.0). Infer převádí překladové příkazy na příkazy pro clang/javac. Konflikty mezi verzemi překladačů mohou vzniknout už na úrovni překladových příkazů (některé nastavení není podporováno/rozpoznáno) nebo až v překládaném kódu (nepřeložitelná konstrukce). Problémy nekompatibility je nutné řešit ručně a může se stát, že nekompatibilita nebude v rozumném měřítku řešitelná. Dalším problémem může být vysoký počet hlášení, z nichž může být vysoké procento falešných, pokud je již software hotový, ověřování hlášení je náročnější a případné úpravy také. Rovněž jednorázový příval velkého množství hlášení k ověření je nepříjemný.

Nasazením Inferu již do vývoje se lze z části výše zmíněným problémům vyhnout. Lze např. zvolit vhodnou verzi překladače a sestavovacího systému s ohledem na Infer (v již hotovém softwaru by se to měnilo těžko). Nastávají zde však nové problémy. Hlavním problémem je automatizace procesu. Pokud by si měl nad každou změnou programátor pouštět Infer ručně, ztratila by se tím spousta času, proto je automatizace velkou výhodou. S automatizací však přicházejí další problémy - uchovávání chyb, odstraňování falešných hlášení, odstranění zastavení analýzy, způsob hlášení výsledků, integrace do verzovacího systému atd. Těmito problémy se zabývají další kapitoly.

Kód firmy CAMEA Tento software byl složen ze čtyř částí - detektor vzorů v C, detektor vzorů ve vhdl, ovladač pro DMA v C a detektor vzorů v C++ (podrobnější informace nejsou uvedeny z důvodu autorských práv). Bylo možné zkontrolovat pouze části napsané v C nebo C++. Prvním krokem bylo nainstalování potřebných závislostí a jejich správných verzí, aby bylo možné projekt sestavit. Návod na sestavení byl přiložen pouze u jediné části v podobě cmake specifikace. Po úspěšném sestavení bylo potřeba získat překladové příkazy. Jako sestavovací systémy byly použity make a cmake, které Infer podporuje. U sestavení pomocí cmake se ale Infer nedostal k překladovým příkazům, proto bylo nutné je exportovat do souboru ve formátu .json a následně je předat Inferu. U ostatních částí dokázal Infer exportovat příkazy přímo z provedení make. Dalším problémem byla nekompatibilita překladačů. Bylo nutné provést úpravy přímo v kódu. V C++ kódu nastal problém se speciální konstrukcí přetěžování operátorů, kterou interní verze clang z Inferu nepodporuje. Toto bylo vyřešeno ruční úpravou zdrojových souborů. V C kódu nastal problém s makrem ze standardní knihovny, které využívalo konstrukci asm_goto, kterou interní verze clangu nepodporovala. Toto bylo vyřešeno nahrazením tohoto makra za makro prázdné. Při úpravách zdrojových souborů je důležité, aby možné změny neodstranily nějaké chyby. To se zpravidla nestane, avšak pokud je nahrazen nebo vymazán skok (zde nastalo) nebo konstrukce řídící tok programu, tak se mohla nějaká chyba odstranit. V tomto případě bylo vymazaní jediným řešením. Případné přidané chyby je možné poté pomocí zpětného sledování chyb vyloučit. V projektu z nalezených problémů bylo prokázáno pouze několik, jednalo se o chyby typu přetečení proměnné. Podrobnější informace o analýze jsou uvedeny v hlášení (odkaz v Přiložených materiálech, hlášení je psáno v anglickém jazyce).

Kodek CCSDS formátu v projektu Aquas Tento software byl napsán v čistém jazyce C za použití pouze standardních knihoven, proto nebyl problém s kompatibilitou překladačů. Jako sestavovací nástroj byl použit make, který Infer podporuje, proto extrahování překladových příkazů proběhlo také bez problémů. Software je stále ve vývoji a tomu také odpovídalo množství a typ chyb. Byly zde nalezeny chyby typu chy-

bějící uzavření souboru nebo úniky paměti. Při analýze se objevila skutečnost, že Infer nesprávně pracuje s vícerozměrnými poli, které jsou předávány ve formátu int array[n] (při přístupu k poli Infer hlásí přístup přes NULL ukazatel). Při předávání ve formátu int* array už falešné hlášení nevznikne. Jelikož se jednalo o relativně krátký software, byla navrhnuta možnost zakomponovat Infer do vývoje s tím, že by se při každé změně kontroloval celý software (celková analýza nepřesáhne pár minut). Při dalším zkoumání softwaru se však objevil problém, který komplikoval celkovou analýzu. Jelikož tento software musí být rychlý je zde využito direktiv preprocesoru, které na základě hodnot maker (představující nastavení) odstraňují nepotřebný kód. Jelikož toto odstranění proběhne v překladači dříve než vygenerování mezikódu pro Infer, není možné zkontrolovat části, které podle aktuálního nastavení budou smazány. Prvním logickým krokem bylo přepsání softwaru tak, aby nebylo použito direktiv, ale pouze klasického větvení. Tím se neovlivní funkčnost programu, pouze jeho efektivita. Takto upravené zdrojové kódy bylo možné zkontrolovat celé. Bohužel je tato úprava nepřípustná a bylo by nutné ji dělat jenom interně pro potřeby Inferu (bez ovlivnění skutečných souborů). Další možností je využití skriptu, který bude v hlavičkovém souboru představujícím nastavení měnit makra (parametry) a analýza bude pro každou nebo vybranou možnost vstupních parametrů probíhat znovu. Zde bude také nutnost evidovat již nahlášené chyby a filtrovat chyby nové. Navíc čas potřebný k analýze by při vysokém počtu možností nastavení byl neúnosný. Tento problém se aktuálně stále řeší.

Jádro platformy Arrowhead Na této platformě později poběží software od firmy CAMEA, proto logickým krokem bylo i zanalyzování této platformy. Zdrojové soubory jsou napsány v Javě, tudíž je možné použít Infer. Jako sestavovací nástroj je použit Maven, který je taktéž podporován Inferem. Bohužel Infer nebyl schopen ze sestavovacího procesu získat překladové příkazy a nebylo toho docíleno ani ručně. Proto se upustilo od analýzy softwaru jako celku. Následoval pokus analyzovat platformu po částech s vlastními překladovými příkazy. Při překladu však bylo zjištěno, že je celá platforma napsána pro javac verzi 11+ a tudíž nemohl být zanalyzován Inferem, který používá verzi 8. Ve zdrojových kódech se vyskytovaly speciální typy tříd, které verze 8 nepodporuje. Přepsání zdrojových souborů by nebylo v rozumném čase zvládnutelné, proto nemohl být software zanalyzován.

Ostatní projekty Dalšími analyzovanými programy, zejména pro získání dat k vytvoření statistik ohledně Inferu, byly projekty z předmětu IFJ, IOS, IJC, IZP a jiné. A také soukromé projekty a testy vytvořené přímo za účelem otestování Inferu.

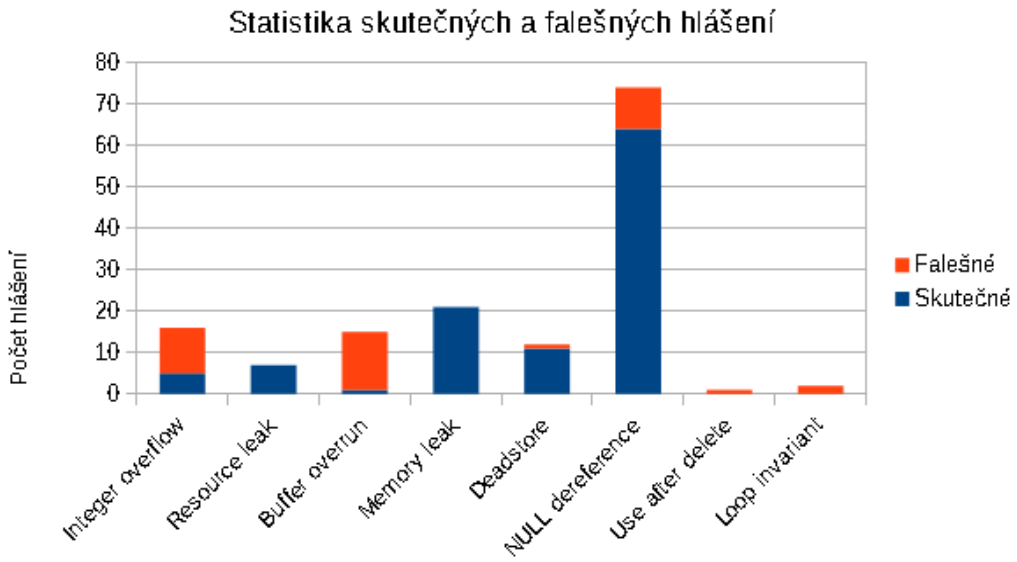
4. Přizpůsobení Inferu na kontrolu assertů

V průběhu praxe vyvstal požadavek od firmy Honeywell ohledně kontroly assert maker pomocí statické analýzy. Asserty v jejich kódech jsou často automaticky generovány z formálních požadavků. Pomocí statické analýzy se mělo zjistit, zda některý assert může být porušen či nikoliv. Infer ale není na jejich kontrolu určený a proto je bere pouze jako obyčejné makro a jejich porušení se nijak neprojeví ve výsledcích analýz. Firma Honeywell to ošetřila makrem, které přepíše každý assert tak, že místo ukončení programu při porušení podmínky dojde k chybě (odkaz přes NULL pointer) a to je Infer schopen detekovat a nahlásit. Pro Honeywell bylo také důležité, aby nástroj dobře škáloval a nehlásil falešné porušení (nebo co nejméně), proto testovali celou sadu statických analyzátorů. Podle výsledků by Infer splňoval schopnost škálování, ale udával velké množství falešných hlášení. Naším úkolem bylo jejich počet snížit.

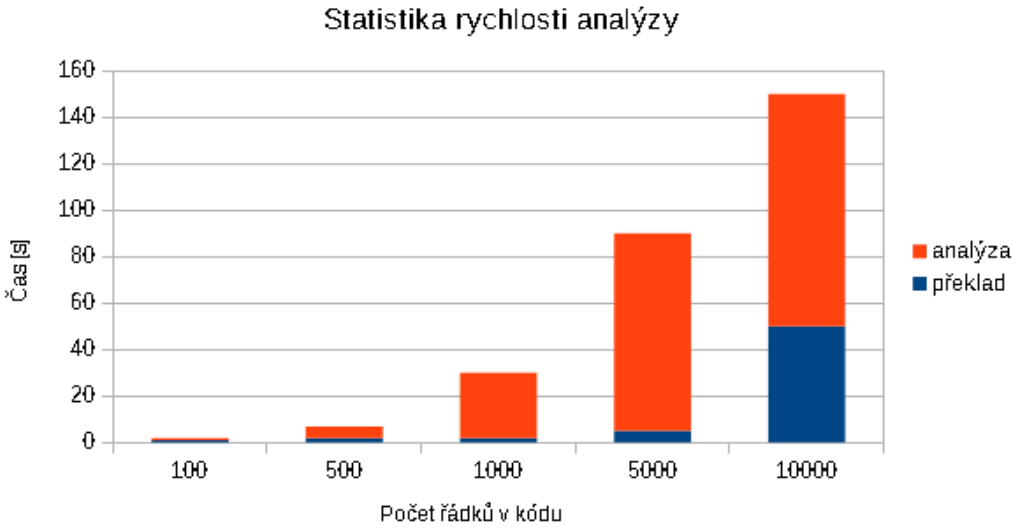
Provedly se dvě úpravy v jejich postupu. První byla úprava makra, kde se odstranila menší chyba, která mohla zapříčinit chybné vyhodnocení podmínky. A druhou úpravou bylo nahrazení chyby jiným typem, konkrétně přístupem za hranice pole, protože analyzátor inferbo sleduje možné hodnoty proměnných přesněji než analyzátor bi-abduction, který hodnoty poměrně hodně nadaproximovává. Díky tomu však inferbo škáluje hůře než bi-abdukce, nicméně i tak inferbo splňovalo podmínku škálování firmy Honeywell.

5. Výsledky ověření vlastností

Následující grafy ukazují sumarizované výsledky analýz. Jedná se pouze o orientační data, jelikož dané vlastnosti závisí na typu softwaru. Rovněž jsou do statistik přidány výsledky z analýz školních projektů a softwarů, které ještě neprošly řádným testováním. Výsledky jsou naměřeny na běžném osobním notebooku - procesor Intel Celeron 1,1GHz (2 jádra), RAM 3GB.



Obrázek 1. Graf výskytu jednotlivých typů hlášení v reálném softwaru a znázornění skutečných chyb



Obrázek 2. Graf doby trvání analýzy reálného softwaru

6. Integrace nástroje Infer s verzovacím systémem Git

Při nasazení Inferu již při vývoji je důležité, aby Infer měl přístup k verzím kódu a jeho změnám. Logickým řešením se jeví zakomponovaní Inferu do verzovacího nástroje tak, aby při každé akci (např. commit, push, merge atd.) byla spuštěna analýza. Při každé změně může analýza probíhat na celém kódu nebo pouze na změněných částech. Krátký kód (do cca. 10 000 řádku, samozřejmě záleží na typu kódu) je možné kontrolovat celý v řádech jednotek minut. U objemnějších kódů toto není možné a je nutné analyzovat pouze změněné části.

Průběžnou analýzou je možné odstranit problém, který vzniká při analýze celého softwaru najednou - velké množství hlášení ke kontrole a tím pádem velká časová zátěž programátora. Má-li však programátor téměř okamžitou zpětnou vazbu, je kontrola hlášení mnohem rychlejší. Jelikož kontrolovat kód, který programátor sám nedávno napsal je mnohonásobně jednodušší než kontrolovat hlášení kódu, který už není v živé paměti programátora. Průběžnou analýzou se také množství chyb rozloží a je lépe zvládnutelné. Další problém, který lze z části eliminovat je zastavení analýzy při nalezení chyby, zejména u analyzátoru bi-abduction. Aby analýza mohla pokračovat je nutné chybu odstranit a poté znovu zanalyzovat. Pokud by se však měl analyzovat celý software, bylo by to časově nevýhodné. Proto je lepší zanalyzovat pouze část, ve které se chyba objevila (to už je v průběžné analýze zařízeno) a provést změnu do kódu

pouze po odstranění/vyloučení toho hlášení. Pokud je však prokázáno, že se nejedná o chybu, odstranění zastavení analýzy je složitější a věnuje se tomu následující kapitola.

Integrace Inferu do Gitu je zařízena pomocí Git hooks. Jedná se o skripty, které jsou zařazeny ve složce .Git/hooks v každém repozitáři a jsou podle potřeby spouštěny před nebo po zvolené akci (commit, push, merge atd.). U tohoto konkretního řešení se používá pre-commit hook. Na rozdíl od nasazení ve Facebooku se kontrolují pouze změněné soubory (ve Facebooku se kontrolují změněné funkce a funkce na nich závislé). Jako změny se počítají soubory, které byly upraveny od posledního commitu. Seznam změn lze získat přímo pomocí Gitu. Před provedením každého commitu je spuštěna analýza a po výpisu hlášení dokončen commit. Plánuje se také doplnění skriptu o jednoduché rozhraní, které dokončí commit pouze v případě, že změny neobsahují žádné hlášení nebo když je ověřeno, že hlášení jsou falešná. Kvůli falešným hlášením je nutné mít uložený seznam všech dřívějších hlášení, které slouží k filtrování výstupu analýzy (filtrování je již zabudované v Inferu), aby se již vyloučená hlášení znovu neobjevovala. Tento seznam je uložen ve složce infer-out/ v repozitáři. Jsou zde také uloženy soubory potřebné pro Infer a je nutné je udržovat aktuální vzhledem ke commitu, tj. musí být sledovány Gitem. Nevýhodou je však nutnost pravidelně analyzovat celý software, jelikož změny mohou způsobit chybu i v jiných souborech. Celková analýza pak může být spuštěna například přes noc.

7. Odstranění zastavení analýzy u analyzátoru bi-abduction

Zatím jedinou experimentálně zjištěnou metodou na odstranění zastavení analýzy u analyzátoru bi-abduction je uzavření místa výskytu do konstrukce if-else s předem neznámou podmínkou (neznámou v tom smyslu, že ji Infer nedokáže předem vyčíslit). Kvůli vnitřní implementaci Inferu se analýza zastaví pouze v dané větvi programu. Každý příkaz je však možné uložit do konstrukce if-else, která nebude mít vliv na funkčnost programu a to tak, že se jako podmínka použije návratová hodnota funkce, u které má Infer přístup pouze k deklaraci. Infer tak předpokládá libovolnou návratovou hodnotu a tudíž musí zkontrolovat obě větve. Funkce bude implementována tak, že vždy bude vracet nenulovou hodnotu (pro jazyk C se hodnota vyhodnotí jako pravda) a tím zaručí, že se při vykonávání programu příkaz nepřeskočí. Jelikož však Infer vidí danou konstrukci jako přeskočitelnou, mohou vzniknout další falešná hlášení. Když se však spustí sledování chyby, je možné určit, zda chyba vznikla touto úpravou. Při úpravě zdrojového souboru je však nevhodné dělat takovéto úpravy přímo v repozitáři. Nejvhodnější variantou by bylo vytvoření lokální kopie určené pouze pro analýzu, která by se podle potřeby aktualizovala. Celý tento proces je plánováno vytvořit jako plně automatizovaný a zakomponovaný do integrace Inferu s Gitem.

8. Závěr

Tato práce se zabývala ověřením vlastností statického analyzačního nástroje Facebook Infer na reálném softwaru. Z výsledků vyplývá, že Infer je univerzálním nástrojem na analýzu zdrojových kódu v jazycích C, C++, Objective-C a Java. Jeho použití je velice jednoduché, velmi dobře škáluje a je také vhodný na nasazení již do vývoje. Problémem bývá zpřístupnění překladových příkazů, kompatibilita vnitřního překladače a množství falešných hlášení, to však lze z části vyřešit nasazením již do vývoje. Integrace Inferu s verzovacím systémem Git může usnadnit nasazení Inferu jako běžné části vývojového procesu softwaru ve firmách. Integrace je však stále ve vývoji a plánují se výše zmíněná rozšíření. Do budoucna by také bylo možné rozšířit integraci o více statických analyzačních nástrojů.

Poděkování

Děkují vedoucímu své projektové praxe Prof. Ing. Tomáši Vojnarovi Ph.D. za odborné vedení, cenné rady, podněty a připomínky, dále Ing. Tomáši Fiedorovi, Mgr. Tomáši Kratochvílovi, Ing. Martinu Musilovi, Ing. Davidu Bařinovi Ph.D. za jejich zpětnou vazbu k výsledkům analýz.

Zdroje

[1] developers.redhat.com/blog/2017/09/06/ continuous-integration-a-typical-process/
[2] fbinfer.com/
[3] pajda.fit.vutbr.cz/xmarci10/fbinfer_concurr ency/-/wikis/L2D2:-A-Low-Level-Deadlock-Detector
[4] github.com/harmim/infer/wiki/Atomer:-Ato micity-Violations-Analyser
[5] www.sstic.org/media/SSTIC2019/SSTIC-actes/journey-to-a-rte-free-x509-parser/SSTIC2019-Article-journey-to-a-rte-free-x509-parser-ebalard_mouy_benadjila_3cUxSCv.pdf
[6] fbinfer.com/docs/analyzing-apps-or- projects.html
[7] research.fb.com/wp-content/uploads/2016/11/publication00124_download0001.pdf
[8] fbinfer.com/docs/separation-logic-and-bi-abduction.html