VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE VÝSLEDKŮ STATICKÉHO POROVNÁVÁNÍ SÉMANTICKÉ EKVIVALENCE RŮZNÝCH VERZÍ SOFTWARE V JAZYCE C

VISUALISATION OF STATIC COMPARISON OF SEMANTIC EQUIVALENCE OF DIFFERENT VERSIONS OF SOFTWARE WRITTEN IN C

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ PETR**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. TOMÁŠ VOJNAR, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144949

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Petr Lukáš**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Vizualizace výsledků statického porovnávání sémantické ekvivalence různých verzí software v jazyce C**
Kategorie:	Analýza a testování softwaru
Akademický rok:	2022/23

Zadání:

1. Seznamte se s nástrojem DiffKemp pro automatickou statickou analýzu sémantických rozdílů mezi verzemi jádra Linuxu či jiných, typicky systémových programů napsaných v jazyce C. Zaměřte se hlavně na informace, které nástroj poskytuje na svém výstupu.
2. Prozkoumejte existující přístupy pro vizualizaci rozdílů ve zdrojových kódech a pro navigaci v projektech napsaných v jazyce C.
3. Navrhněte způsob efektivního a kompaktního zobrazení výstupu z nástroje DiffKemp tak, aby zachycoval jak nalezené rozdíly, tak i vztahy (zásobníky volání) mezi analyzovanými funkcemi.
4. Implementujte Vámi navržené řešení pomocí vhodné technologie (web, textové uživatelské rozhraní, ...).
5. Navrhněte způsob vyhodnocení kvality Vašeho řešení a realizujte ho. Diskutujte zjištěné výsledky.

Literatura:

- V. Malik and T. Vojnar. Automatically Checking Semantic Equivalence between Versions of Large-Scale C Projects. In Proc. of the IEEE International Conference on Software Testing, Verification and Validation 2021 -- ICST 2021, IEEE, 2021.
- The official website of DiffKemp: https://github.com/viktormalik/diffkemp
- The Elixir Cross Referencer: https://github.com/bootlin/elixir.

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání a alespoň začátek práce na bodě třetím.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Vojnar Tomáš, prof. Ing., Ph.D.**
Konzultant:	Ing. Viktor Malík
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	3.11.2022

Abstrakt

Cílem této práce je vytvořit přehlednější prezentaci výsledků nástroje DiffKemp sloužícího pro statickou analýzu sémantických rozdílů ve velkých projektech napsaných v jazyce C. V současnosti DiffKemp zobrazuje všechny informace o nalezených rozdílech v nestrukturované textové podobě, což je pro uživatele mnohdy nepřehledné. Pro vyřešení tohoto problému byl v této práci vytvořen nový výstup nástroje DiffKemp, který poskytuje výsledky v serializované podobě pomocí formátu YAML. Tento výstup je následně zpracován a zobrazen pomocí nově vytvořeného prohlížeče výsledků, realizovaného jako webová aplikace pomocí knihovny React, frameworku Bootstrap a balíčku react-diff-view. Dále se práce zaměřila na poskytnutí uživateli dodatečného kontextu ve formě zdrojových kódů analyzovaných funkcí a usnadnění navigace a orientace v nalezených rozdílech a k nim poskytovaných informací jako jsou zásobníky volání. Provedené srovnání nově vytvořeného prohlížeče ukázalo, že usnadňuje uživateli oproti původnímu řešení rozpoznat změny v poskytovaných zásobnících volání a umožňuje mu rychlejší navigaci ve výsledcích a také mezi vztahy nalezených rozdílů a analyzovaných částí.

Abstract

The aim of this thesis is to create a more comprehensive presentation of results of the DiffKemp tool, which is used for static analysis of semantic differences in large projects written in C. Currently, DiffKemp displays all information about the found differences in an unstructured text which is often confusing for users. To solve this problem, a new output of the DiffKemp tool was created in this thesis, which provides the results in a serialized form using the YAML format. This output is subsequently processed and displayed using a newly created results browser, implemented as a web application using the React library, the Bootstrap framework, and the react-diff-view package. In the browser, we focus on providing an additional context in the form of source codes of the analyzed functions, and on facilitation of navigation and orientation in the found differences as well as in the provided information such as the call stacks. A comparison of the newly created browser with the original solution has shown that it is easier for the user to recognize changes in the provided call stacks and that the new browser allows him to navigate faster in the results as well as between relationships of the found differences and analysed parts.

Klíčová slova

DiffKemp, statická analýza sémantických rozdílů, vizualizace rozdílů, prohlížeč výsledků, React, zásobník volání

Keywords

DiffKemp, static analysis of semantic differences, visualization of differences, result viewer, React, call stack

Citace

PETR, Lukáš. *Vizualizace výsledků statického porovnávání sémantické ekvivalence různých verzí software v jazyce C.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Tomáš Vojnar, Ph.D.

Vizualizace výsledků statického porovnávání sémantické ekvivalence různých verzí software v jazyce C

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Tomáše Vojnara, Ph.D. Další informace mi poskytl Ing. Viktor Malík. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Lukáš Petr
5. května 2023

Poděkování

Rád bych poděkoval konzultantovi, panu Ing. Viktorovi Malíkovi, za ochotu, rady, podporu a věnovaný čás. Také děkuji svému vedoucímu prof. Ing. Tomáši Vojnarovi, Ph.D.

Obsah

Kapitola 1

Úvod

Zdrojové kódy programů se často upravují a v některých případech je žádoucí, aby význam (sémantika) některých částí programu zůstala zachována i po úpravách. Nástroj DiffKemp je jedním z nástrojů, který umožňuje zkontrolovat, zda nedošlo ke změně významu vybraných částí programu. Jedná se o automatický *statický analyzátor sémantické ekvivalence*, který slouží pro analýzu projektů napsaných v jazyce C. Jeho hlavní zaměření je na jádro Linuxu firmy Red Hat. Pro dvě verze stejného programu umožňuje nástroj porovnat vybrané funkce, které kontroluje včetně volaných funkcí, a ověřit zda jejich sémantika zůstala zachována. V případě, že tomu tak není a jsou nalezeny rozdíly, které se často vyskytují v některé z volaných funkcí, umožňuje nástroj tyto rozdíly uživateli zobrazit. Pro porovnané funkce jsou nalezené rozdíly ukládány do textových souborů, ve kterých jsou uvedeny funkce s nalezeným rozdílem, vztahy porovnávaných a rozdílných funkcí (zásobníky volání) a samotné rozdíly funkcí v textovém formátu.

Způsob prezentace nalezených rozdílů, který nástroj DiffKemp aktuálně poskytuje, není úplně uživatelsky přívětivý, jelikož se jedná o textové soubory. Protože ani existující aplikace by nebyly úplně dostačující pro zobrazování nalezených rozdílů se všemi poskytovanými informacemi, zabývá se tato práce návrhem a implementací nového způsobu prezentace v podobě *prohlížeče výsledků*, jehož cílem je zlepšit uživatelskou přívětivost. Tento prohlížeč bude realizován moderním způsobem jako webová aplikace.

Pro získání informací o nalezených rozdílech bude nutné prohlížeč nejprve propojit s nástrojem DiffKemp. Poněvadž současný výstup nástroje DiffKemp poskytující výsledky analýzy není úplně nejlepší pro další zpracování, bude potřeba nejdřív vytvořit serializovanou podobu těchto výsledků, kterou bude možné potom jednoduše zpracovat a využít za účelem vizualizace.

K přehlednému zobrazení výsledků bude nezbytné navrhnout a realizovat uživatelské rozhraní, které poskytne uživateli dostatečný kontext k nalezeným rozdílům v podobě zdrojových kódů analyzovaných funkcí. K tomu bude nutné vybrat vhodnou knihovnu, s jejíž pomocí bude možné toto zajistit. Také bude potřeba se zaměřit na navigaci ve výsledcích proto, aby byla snadnější a umožnila jednodušeji zjistit, které porovnané funkce jsou ovlivněny konkrétním nalezeným rozdílem. Pozornost bude třeba také věnovat zobrazení zásobníků volání, aby nebylo zbytečně složité. Tímto by mohla nová podoba prezentace výsledků také ulehčit uživateli vyhodnocení závažnosti nalezených rozdílů.

Zbytek této práce je strukturován následovně. Kapitola 2 slouží k seznámení se s nástrojem DiffKemp – s tím, jak funguje, z jakých částí se skládá a také s informacemi, které poskytuje na svém výstupu. Následující Kapitola 3 se zaměřuje na specifikaci požadavků pro vytvoření řešení, průzkumem existujících aplikací sloužících pro vizualizaci rozdílů a na-

vigaci v kódu, popisem použitých technologií a výběrem knihovny pro zobrazování rozdílů. Další Kapitola 4 se zabývá návrhem rozšíření – integrací prohlížeče do nástroje DiffKemp, návrhem uživatelského rozhraní a návrhem nového výstupu nástroje, který by usnadnil zpracování výsledků pro vizualizaci. V Kapitole 5 je popsána implementace řešení, která se skládá z vytvoření nového výstupu nástroje DiffKemp sloužícího k předání výsledků prohlížeči, přípravy výsledků pro zobrazení a vytvoření samotného prohlížeče výsledků realizovaného v podobě webové aplikace pomocí knihovny React, frameworku Bootstrap a vybrané knihovny pro zobrazování rozdílů. Předposlední Kapitola 6 se zabývá vyhodnocením kvality řešení pomocí vytvoření automatizovaných testů pro implementované rozšíření, manuální kontrolou prohlížeče výsledků pro některé předešlé verze jádra Linuxu a srovnáním nové prezentace výsledků s původní. Poslední Kapitola 7 shrnuje provedenou práci a navrhuje možnosti dalšího vývoje na prohlížeči.

Kapitola 2

Nástroj DiffKemp

DiffKemp[1] je nástroj sloužící ke kontrole toho, zda refaktorizací programu nenastala sémantická změna. Je určen pro rozsáhlé projekty napsané v jazyce C, které potřebují být sémanticky stabilní a zpětně kompatibilní napříč verzemi [11]. Pomocí *statické analýzy* dokáže *automaticky* porovnat dvě verze stejného programu a zkontrolovat, zda nová verze programu nemá odlišnou *sémantiku*. V případě, že tato situace nastane, nástroj informuje o nalezených rozdílech.

DiffKemp využívá odlehčený přístup *statické analýzy sémantické rovnosti*, který zvládne porovnat velké množství funkcí (tisíce) v rámci krátkého časového intervalu (minut) s nízkým počtem nesprávně vyhodnocených nerovností [11]. Existují rychlejší nástroje jako např. unixový nástroj `diff`, který ovšem zvládá maximálně jednoduché sémantické změny. Na druhou stranu existují i nástroje založené na formálních metodách, které se zabývají touto problematikou. Vzhledem k výpočetní náročnosti formálních metod nejsou tyto nástroje stavěny na kontrolu velkého množství kódu. Příkladem těchto nástrojů jsou SymDiff [7] nebo LLReve [6].

Pro analýzu pomocí nástroje DiffKemp jsou potřeba dvě verze stejného programu. Kontrola sémantické rovnosti je prováděna po jednotlivých funkcích. Nejprve jsou zdrojové kódy přeloženy do LLVM IR (vnitřní reprezentace LLVM), poté dochází k různým transformacím a vyhodnocení rovnosti funkcí. Tuto ekvivalenci se snaží DiffKemp vyhodnocovat porovnáváním na úrovni jednotlivých instrukcí. Pokud to není možné, snaží se zjistit zda změna odpovídá nějakému vestavěnému vzoru zachovávajícího sémantiku (semantics-preserving change patterns). Algoritmus analýzy rovnosti funkcí je podrobněji popsán v článku [11].

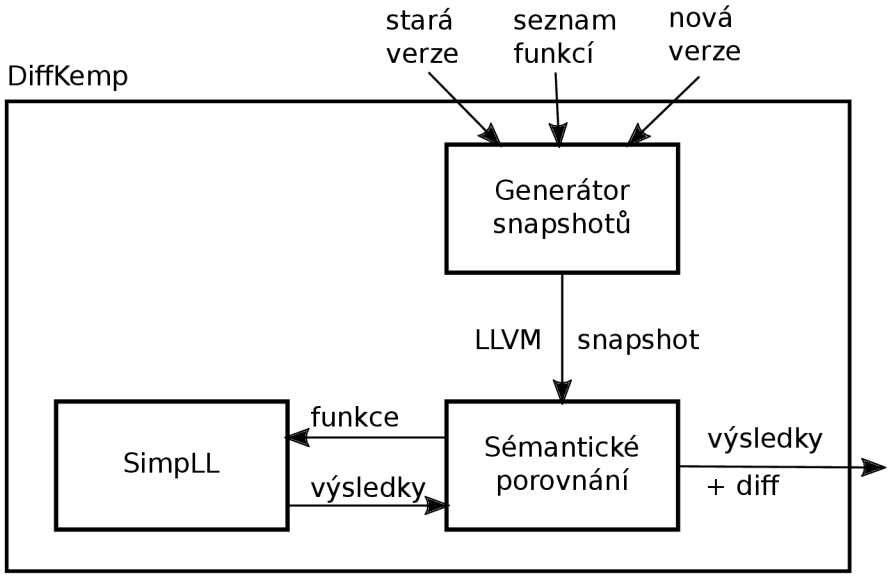
Hlavní zaměření nástroje je na operační systém Red Hat Enterprise Linux (RHEL), jehož jádro obsahuje seznam funkcí, tzv. Kernel Application Binary Interface (KABI) [11]. Sémantika těchto funkcí by měla zůstat stejná v rámci jedné hlavní verze RHEL.

Zbytek této kapitoly popisuje architekturu nástroje DiffKemp. Sekce 2.1 je do této oblasti úvodem, po této sekci nasledují podkapitoly popisující jednotlivé části více do detailu. V Sekci 2.2 je popsáno generování snapshotů, které je nezbytné provést před samotnou analýzou. Další Sekce 2.3 je zaměřena na proces porovnání a vyhodnocení sémantické rovnosti funkcí. Na závěr následuje Sekce 2.4, ve které je popsán aktuální výstup programu.

[1] Nástroj DiffKemp je dostupný z repozitáře `https://github.com/viktormalik/diffkemp`

2.1 Architektura

Tato sekce pojednává o architektuře nástroje DiffKemp, která je znázorněna na Obrázku 2.1. Další sekce se potom zaměřují na jednotlivé části více dopodrobna.



Obrázek 2.1: Obrázek architektury nástroje DiffKemp

Proces analýzy je následující:

- Na začátku je nutné mít dvě verze stejného programu a seznam funkcí, které chceme porovnat, poté můžeme nástroj spustit. V první části je vytvořen z každé verze tzv. snapshot. Více informací o **generování snapshotů** lze najít v Sekci 2.2.
- Následně je možné realizovat **sémantické porovnání**, které je provedeno pro dříve uvedené funkce. Samotné vyhodnocení rovnosti je uskutečněno knihovnou **SimpLL**. Ta poskytuje výsledek, zda je funkce v obou verzích sémanticky ekvivalentní. V případě, že tomu tak není, obsahuje informaci o nalezených rozdílech. Viz Sekce 2.3.
- **Výsledky** a nalezené rozdíly všech vybraných funkcí jsou nakonec **prezentovány** uživateli, viz Sekce 2.4.

Knihovna SimpLL je napsána v jazyce C++ z důvodu výkonnosti. Ostatní části jsou napsány v jazyce Python a slouží jako obal (wrapper) nad touto knihovnou, zde jsou zpracovány vstupy a výstupy z důvodu jednoduchosti [10].

2.2 Generování snapshotů

Tato sekce se zabývá první částí nástroje a to generováním snapshotů.

K analýze je potřeba mít dvě verze programu, které chceme porovnat. V současnosti DiffKemp podporuje pouze projekty napsány v jazyce C. Dále potřebujeme seznam funkcí, pro které chceme zkontrolovat, zda nedošlo ke změně jejich sémantiky. V případě Red Hat Enterprise Linux je tímto seznamem již zmíněný KABI seznam funkcí.

Samotná analýza neprobíhá na úrovni zdrojového kódu, ale na úrovni jazyka LLVM IR. Více o této reprezentaci je popsáno ke konci této sekce.

Jednotlivé verze programu společně se seznamem funkcí musíme předat generátoru, který typicky přeloží zdrojový kód do LLVM IR (vnitřní reprezentace LLVM) a vytvoří tzv. *snapshot*. Ten obsahuje LLVM IR soubory a soubor s metadaty [10], který obsahuje informaci o tom, v kterém souboru se nachází jaká funkce.

K vytvoření snapshotů nám DiffKemp nabízí 3 možnosti [10]:

- Vytvoření z projektů založených na nástroji make.
- Sestavení Linuxového jádra, oproti první možnosti je výhodou, že sestaví pouze soubory obsahující porovnávané funkce.
- Sestavení z již přeloženého projektu do LLVM IR souboru.

Pro správnou funkčnost analýzy programu potřebuje DiffKemp sestavit projekt s ladícími informacemi [10].

LLVM IR

Tato část podkapitoly se zaměřuje na LLVM Intermediate Representation (IR). Na úrovni této reprezentace probíhá analýza programu nástrojem DiffKemp. Zdrojem informací této podsekce byl referenční manuál [9].

LLVM IR kód, který byl navrhnut pro transformaci, analýzu a optimalizaci programů, je způsob reprezentace zdrojového kódu v překladači Clang. Jedná se o typovaný nízkoúrovňový jazyk, který umožňuje zachytit vysokoúrovňové konstrukce různých programovacích jazyků.

Program v LLVM je složen z modulů, které obsahují funkce. Základem jsou instrukce, které jsou v tříadresné formě a skládají se z názvu operace a operandů. Operandem může být lokální proměnná, globální proměnná nebo volání funkce. Výsledek každé instrukce je uložen do nově vytvořené lokální proměnné (princip Static Single Assignment) [11].

Funkce reprezentuje graf toku řízení (CFG), který je tvořen základními bloky. Ty jsou složeny z posloupnosti jednotlivých instrukcí a ukončeny instrukcí větvení nebo návratu z funkce. Tyto ukončující instrukce specifikují následující základní blok.

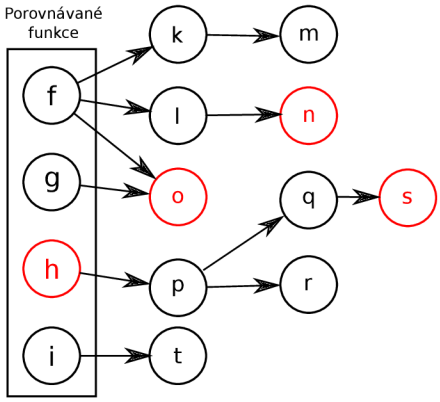
2.3 Sémantické porovnání

Vytvořené *snapshoty* je možné *sémanticky* porovnat. Vyhodnocení je provedeno *statickou analýzou*, tedy zkoumáním vlastností programu bez jeho spuštění [15]. Analýza je provedena po jednotlivých funkcích. Pro výběr toho, co se má porovnat, nástroj poskytuje dvě možnosti:

- První varianta umožňuje porovnat všechny funkce ze seznamu, který byl specifikován při generování snapshotu.
- Druhou možností je zvolit si pouze jednu z těchto funkcí.

Po spuštění analýzy, která potřebuje cestu k adresářům s vytvořenými *snapshoty*, je vyhodnocována sémantická rovnost porovnávaných funkcí a to včetně funkcí přímo či nepřímo volaných.

Na Obrázku 2.2 je uveden příklad grafu volání funkcí s vyznačenými porovnávanými a sémanticky různými funkcemi. Můžeme vidět, že jsou porovnávány funkce `f`, `g`, `h` a `i`.



Obrázek 2.2: Příklad grafu volání, který zobrazuje porovnávané a sémanticky různé funkce. V obdélníku jsou znázorněny funkce, které jsou porovnávány. Hrany směřují z funkcí do funkcí jimi volaných. Červeně jsou označeny funkce, ve kterých je nalezen sémantický rozdíl.

Rozdíl v porovnávané funkci bývá ve skutečnosti většinou nalezen v některé z volaných funkcí – např. funkce `f` obsahuje sémantické rozdíly ve funkcích `n` a `o`. Může se stát, že pro jednu porovnávanou funkci je nalezen rozdíl ve více volaných funkcích nebo pouze v jedné – např. funkce `g` se liší pouze ve funkci `o`. Taktéž se stává, že rozdíl je nalezen v některé volané funkci i v samotné porovnávané funkci – např. funkce `h` se liší ve funkci `s` a v sobě samé. Je možné, že funkce s nalezeným rozdílem, je volána z více porovnávaných funkcí – např. funkce `o` je volána z funkcí `f` a `g`. Rozdíl nemusí být nalezen jenom ve funkci, ale také v datovém typu používaném funkcí nebo v makru.

Samotná analýza ekvivalence je provedena knihovnou SimpLL, která vrací informaci o rovnosti jednotlivých funkcí. V případě Obrázku 2.2 by knihovna vrátila, že funkce `n`, `o`, `h` a `s` jsou sémanticky neekvivalentní (`not-equal`) a ostatní ekvivalentní (`equal`). Tedy i funkce `f` a `g` by byly označeny jako ekvivalentní, vyhodnocení porovnávaných funkcích vzhledem i k volaným je provedeno později.

Získané výsledky z knihovny jsou shromažďovány (viz Sekce 2.3.3) a je z nich sestaven *graf volání*, ze kterého je vyhodnocena ekvivalence porovnávaných funkcí (nyní již i vzhledem k volaným). V případě Obrázku 2.2 by bylo vyhodnoceno, že ve funkcích volaných funkcemi `f`, `g` a `h` byl nalezen rozdíl.

Nakonec jsou výsledky prezentovány uživateli. Poskytují informaci o funkcích s nalezeným rozdílem pro každou porovnávanou funkci. Prezentace výsledků je popsána v Sekci 2.4.

2.3.1 Knihovna SimpLL

Knihovna SimpLL[2], která provádí analýzu programu, je jádrem celého nástroje [10]. Jejím cílem je zjistit zda *syntaktická změna* způsobila změnu v *sémantice*.

Tato knihovna je volána pro každou porovnávanou funkci. Jejím vstupem je jméno analyzované funkce a LLVM soubory (soubor se starou a s novou verzí) obsahující instrukce této funkce. Následně dochází k samotnému porovnání, které je provedeno na úrovni LLVM instrukcí, tento proces je popsán v samostatné části 2.3.2.

Výsledek analýzy funkce je předán ve formátu YAML. Výstup obsahuje seznam funkcí, v něm se kromě porovnávané vyskytují také volané. Pro každou funkci v seznamu je obsa-

[2]Program simplifier for analysis of semantic difference

žena informace o výsledku, neboli zda je daná stará a nová funkce sémanticky stejná nebo se liší. Kromě toho jsou zde uvedeny další údaje pro obě verze funkce:

- název (z důvodu toho, že mohlo dojít k jeho změně),
- cesta k souboru a číslo řádku, kde je funkce definována,
- seznam volaných funkcí – jejich název, dále soubor a řádek udávající, kde byla volána.

Výsledek analýzy funkce je uložen v části napsané v Pythonu, kde jsou shromažďovány jednotlivé výsledky. Více o tomto tématu je popsáno v části 2.3.3.

2.3.2 Proces porovnání funkcí

Zdrojem informací následující části, která se zabývá vyhodnocováním rovnosti funkcí, je článek [11]. Proces porovnání je založený na dvou krocích:

1. Nejdříve dochází k předzpracování LLVM reprezentace funkce, jejímž cílem je zjednodušit následné porovnání po jednotlivých instrukcích. Předzpracování je provedeno transformacemi kódu, které nemění sémantiku funkce. Mezi tyto úpravy patří např. propagace konstant a eliminace mrtvého kódu a nepoužitých parametrů.
2. Následně dochází k samotnému porovnání, které běžně probíhá po jednotlivých instrukcích. Kromě toho DiffKemp také obsahuje předdefinované vzory změn zachovávající sémantiku. Mezi tyto vzory patří např.: změna struktury datového typu, přesun části kódu do funkcí, změna hodnot výčtového typu. Pokud změna odpovídá některému ze vzorů, tak je vyhodnocena jako ekvivalentní. V případě, že dochází k volání nějaké funkce, tak se proces analýzy provádí i pro tuto funkci.

2.3.3 Agregace výsledků

Výstup vyhodnocení funkcí, získaný od knihovny SimpLL ve formátu YAML, je následně uložen jako *graf volání* v části, která je napsána v Pythonu. Graf volání je orientovaný graf, jehož uzly představují funkce programu a jeho hrany reprezentují volání funkce [16]. Příklad grafu volání je možné vidět na Obrázku 2.2. V tomto případě jsou v uzlech obsaženy následující informace:

- výsledek vyhodnocení dané funkce,
- pro obě verze funkce (starou i novou) je uchováno (1) jméno funkce, (2) cesta k souboru a (3) číslo řádku, kde je funkce definována,
- ne-funkční rozdíly – seznam objektů obsahující informace o nalezených rozdílech v datových typech a makrech, které jsou používány touto funkcí. V případě datového typu je pro obě jeho verze uchována (1) cesta k souboru a (2) řádek, kde začíná jeho definice. Pro oba typy rozdílů (makra i datové typy) je uchováno jejich jméno a zásobník volání, který začíná používající funkcí.
- Seznam hran. Hrana obsahuje informace o volání funkce: její jméno, řádek a soubor, ve kterém je volána.

Pro porovnávané funkce je sestaven výsledek, který obsahuje seznam funkcí, ve kterých je nalezen rozdíl. Z grafu volání jsou pro tyto funkce sestaveny zásobníky volání. Dále je pro tyto funkce vytvořen syntaktický rozdíl za použití unixového nástroje `diff`.

2.4 Prezentace výsledků

Výsledek analýzy je prezentován jako textový výstup. K dispozici jsou dvě možnosti prezentace:

- První variantou je výpis výsledků na standardní výstup (zejména pro účely ladění).
- Druhou možností je uložení výstupu do adresáře. V tomto případě je pro každou porovnávanou funkci s nalezeným rozdílem vytvořen v dané složce samostatný soubor informující o nalezených rozdílech. Soubor nese název podle porovnané funkce s příponou `.diff`.

V obou případech je uživatel na standardní výstup informován o funkcích, které se nepodařilo porovnat.

```
Found differences in functions called by call_usermodehelper

call_usermodehelper_exec_async differs:
  Callstack (kernel-src/SNAPSHOT-80/):
  call_usermodehelper_setup at kernel/umh.c:601
  call_usermodehelper_exec_work at kernel/umh.c:385
  call_usermodehelper_exec_async at kernel/umh.c:190

  Callstack (kernel-src/SNAPSHOT-147/):
  call_usermodehelper_setup at kernel/umh.c:622
  call_usermodehelper_exec_work at kernel/umh.c:389
  call_usermodehelper_exec_async at kernel/umh.c:194

  Diff:
  *************** static int call_usermodehelper_exec_async(
     void *data)
  *** 102,107 ***
        sub_info->pid = task_pid_nr(current);
  !     if (sub_info->file)
            retval = do_execve_file(sub_info->file,
                        sub_info->argv, sub_info->envp);
  !     else
            retval = do_execve(getname_kernel(sub_info->path),
  --- 104,111 ---
        sub_info->pid = task_pid_nr(current);
  !     if (sub_info->file) {
            retval = do_execve_file(sub_info->file,
                        sub_info->argv, sub_info->envp);
  !         if (!retval)
  !             current->flags |= PF_UMH;
  !     } else
            retval = do_execve(getname_kernel(sub_info->path),
```

Výpis 2.1: Příklad prezentace výsledků nástroje DiffKemp. Možné je vidět, že byla porovnána funkce `call_usermodehelper`, ve které byl nalezen sémantický rozdíl. Ten

byl objeven ve funkci `call_usermodehelper_exec_async`. Dále je možné vidět jednotlivé zásobníky volání a nakonec textový rozdíl dané (rozdílné) funkce.

Na Výpisu 2.1 je možné vidět konkrétní příklad prezentace výsledků nástroje Diffkemp. Výsledek každé porovnané funkce, ve které byl nalezen rozdíl, obsahuje seznam funkcí s nalezenými sémantickými rozdíly. Pro každou takovou funkci jsou vypsány dva zásobníky volání (posloupnosti volání funkcí od porovnávané funkce k funkci obsahující sémantický rozdíl) – jeden zásobník pro starou verzi programu, druhý pro novou. Zásobník volání obsahuje samotná volání, která obsahují:

- jméno volané funkce (v případě maker a datových typů je za jejich názvem v závorce uvedeno, že se jedná o typ/makro),
- relativní umístění souboru (vzhledem ke kořenovému adresáři analyzovaného projektu), ve kterém je funkce volána a
- číslo řádku, na kterém je volána.

Zásobníky mohou být různé a to například z důvodu:

- přemístění kódu funkce do jiného souboru,
- úpravy souboru obsahujícího funkci (přidání/odstranění některých částí) – v nové verzi je kvůli tomu následující funkce volána na jiném řádku než ve staré verzi,
- volání jiných funkcí (respektive použití jiných maker) v nové verzi programu.

Zásobníky se také mohou lišit v počtu volaných funkcí (maker).

Ve zbytku příkladu je možné vidět samotný syntaktický rozdíl, jenž způsobil sémantickou změnu.

Kapitola 3

Návrh nového způsobu prezentace výsledků

Cílem této práce je vytvořit nástroj, který bude přehledně prezentovat výsledky nástroje DiffKemp. První Sekce 3.1 této kapitoly se zabývá specifikací požadavků, které by mělo splňovat výsledné řešení. V následující Sekci 3.2 jsou probrány existující aplikace pro zobrazování rozdílů v textu a navigaci v projektech, tyto aplikace splňují některé aspekty nezbytné pro výsledné řešení. Další Sekce 3.3 se zabývá webovými technologiemi, jelikož bylo zvoleno nástroj realizovat jako webovou aplikaci. Poslední Sekce 3.4 se zabývá knihovnami pro zobrazování rozdílů v textu a výběrem jedné z nich pro použití ve výsledném řešení.

3.1 Specifikace požadavků

Pro vytvoření správného softwarového produktu je nutné si nejprve ujasnit, co od výsledného řešení očekáváme, proto se tato sekce zabývá specifikací požadavků.

Současná prezentace výsledků nástroje DiffKemp, která byla popsána v Sekci 2.4, má pouze textovou podobu, která je poměrně nepřehledná a uživatelsky nepřívětivá. Cílem této práce je vytvořit grafické uživatelské rozhraní – prohlížeč výsledků získané nástrojem DiffKemp.

Základní požadavky

Prohlížeč by měl umožňovat jednoduše procházet jednotlivé výsledky a přehledně je zobrazovat. Prohlížeč by měl také usnadnit práci pro uživatele s vyhodnocením závažnosti nalezených sémantických rozdílů v kódu a to poskytnutím dostatečných informací a kontextu k nalezeným rozdílům. Seznam všech požadavků zachycuje Tabulka 3.1.

Základní práce s prohlížečem by měla spočívat v tom, že uživatel si zobrazí seznam funkcí, ve kterých je nalezen sémantický rozdíl pro jím udržovanou funkci (porovnávanou – např. KABI funkci) mezi dvěma verzemi projektu. Nechá si zobrazit jednotlivé rozdíly a popřípadě kontext v jakém je funkce s nalezeným rozdílem volána z jeho funkce. Z těchto informací by měl být schopen vyhodnotit vliv změn na chování porovnávané funkce.

Možná budoucí rozšíření

Následují další požadavky, které by mohlo výsledné řešení v budoucnosti splňovat:

- Prohlížeč umožňuje zobrazit seznam všech funkcí, ve kterých byl nalezen rozdíl.

Tabulka 3.1: Seznam základních požadavků pro nový způsob prezentace výsledků

Číslo	Popis
1	Prohlížeč umí zobrazit seznam porovnávaných funkcí (jejich jména), které byly vyhodnoceny jako neekvivalentní.
2	Uživatel má možnost si nechat zobrazit seznam funkcí, ve kterých byl nalezen rozdíl, pro jim vybranou porovnávanou funkci.
3	Pro zvolenou funkci obsahující rozdíl u konkrétní porovnávané funkce prohlížeč zobrazuje nalezený *rozdíl* a důležitý *kontext*.
4	Rozdíl (syntaktický rozdíl verzí funkce) je zobrazen ve zdrojovém kódu funkce a je vizuálně odlišen.
5	Kontext k nalezenému rozdílu obsahuje *zásobník volání* – seznam funkcí (jejich názvů) od porovnávané funkce k funkci, ve které byl nalezen sémantický rozdíl.
6	Pro jednotlivé funkce ze *zásobníku volání* umožňuje prohlížeč zobrazit jejich zdrojový kód (účelem je poskytnutí uživateli kontext k nalezenému rozdílu).
7	Zdrojový kód obsahuje zvýrazněnou syntaxi.
8	Ve zdrojovém kódu je označen řádek, na kterém dochází k volání následující funkce (ze zásobníku volání).
9	U zobrazeného zdrojového kódu funkce jsou jednotlivé řádky očíslovány tak, aby odpovídali umístění v souboru.
10	Pro zobrazenou funkci je poskytnuta informace o umístění souboru, ve kterém se funkce nachází (cesta k danému souboru).
11	V případě, že nalezený rozdíl není ve funkci, ale v makru nebo datové struktuře, tak je v prohlížeči tato informace uvedena.

- Při zobrazení funkce, ve které je nalezen rozdíl, prohlížeč umožňuje zobrazit seznam porovnávaných funkcí, ve kterých je nalezen rozdíl také v této funkci. Popřípadě prohlížeč umožňuje přepnout zobrazení nalezeného rozdílu na vizualizaci v kontextu jiné porovnávané funkce.
- Uživatel má možnost „schválení“ (určení závažnosti) nalezeného rozdílu a to buď pro všechny porovnávané funkce nebo pouze pro konkrétní funkci.

3.2 Existující přístupy vizualizace rozdílů a navigace v kódu

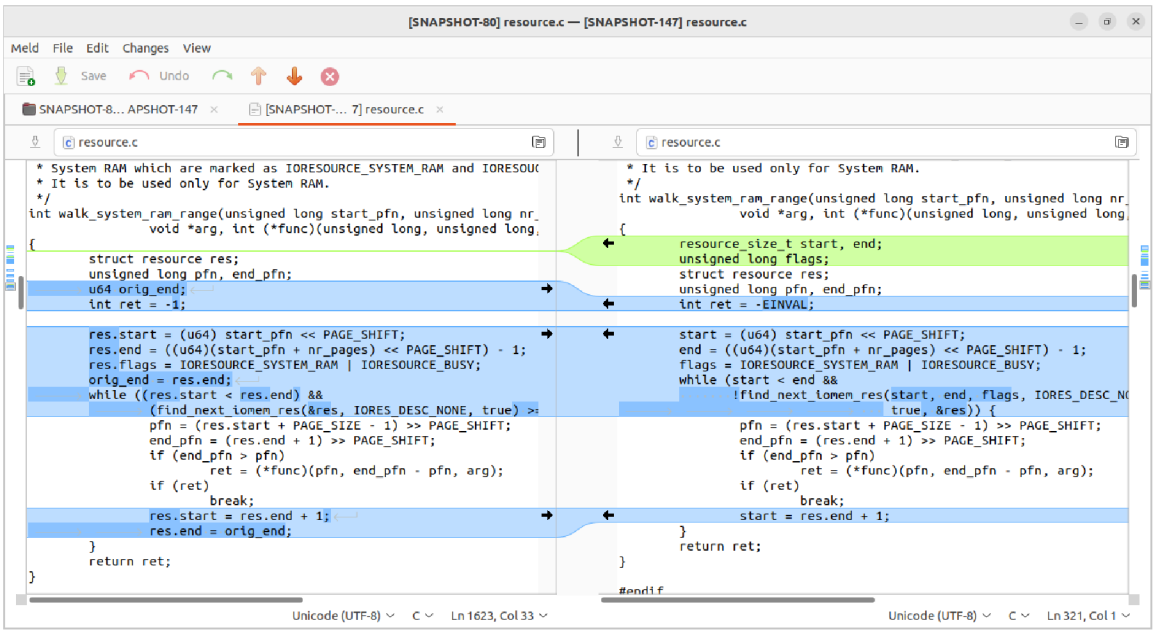
Účelem této sekce je udělat si přehled o existujících aplikacích řešící některé vlastnosti, které by měl mít výsledný prohlížeč.

3.2.1 Vizualizace rozdílů

Pro zobrazení rozdílů existuje například unixový nástroj `diff`, který umožňuje vypsat odlišnosti mezi dvěma textovými soubory. **Výstup** programu je **textový**, obsahuje nalezené rozdíly a k nim informaci o typu (přidání, odebrání, změna) a čísla řádků, kde změna začíná. Pro lidi, kteří s tímto výstupem nepracují často, je tento výsledek poměrně **nepřehledný**. Kromě samotných rozdílů umožňuje vypsat také jejich kontext (řádky kolem rozdílu) a to buď „kopírovaný“ (kontext je identický s originálem – pro každý soubor vlastní text) nebo sjednocený (více kompaktní – jeden text pro oba soubory) [18]. Podle výběru typu kontextu se liší formát výstupního souboru a to především způsobem, jakým se vypisují změny

v souboru, ale také hlavičkami informujících o jednotlivých změnách. Hlavička změny pro „kopírovaný“ kontext obsahuje kromě řádku, kde změna začíná, také informaci, kde změna končí. V případě sjednoceného kontextu obsahuje hlavička místo konce změny počet řádků, kterých se změna týká. Více informací o jednotlivých formátech lze získat z manuálu [2]. Program diff s *kopírovaným* kontextem je aktuálně používán nástrojem DiffKemp pro prezentaci rozdílů v sémanticky různých funkcích.

Nástroje jako **Meld** [19], **KDiff3**[1], **WinMerge**[2] umožňují **vizualizovat** rozdíly mezi soubory **graficky**. Zobrazují oba soubory vedle sebe a jednotlivé změny jsou barevně označeny (např. v nástroji Meld jsou zeleně označeny přidané a odebrané části textu a modře změněné). Jedná se o desktopové aplikace, které kromě zobrazení rozdílů slouží také pro jejich slučování. Na Obrázku 3.1 je ukázka aplikace Meld.



Obrázek 3.1: Obrázek zachycující nástroj Meld [19], který vizuálně odlišuje rozdíly mezi dvěma textovými soubory.

Další přístup poskytuje GitHub[3], webová služba pro hostování projektů, která umožňuje zobrazit rozdíly mezi jednotlivými revizemi (commity) programu. Toto zobrazení je podobné zmíněným desktopovým aplikacím, ale kromě zobrazení souborů se změnami vedle sebe je umožňuje také zobrazit v jednom sloupci (sjednocený pohled).

3.2.2 Navigace v kódu

K navigaci v kódu slouží např. **Elixir** [1]. Jedná se o nástroj, který umožňuje procházet ve webovém prohlížeči zdrojové soubory jednotlivých verzí projektu napsaného v jazyce C nebo C++. Zdrojový kód, který zobrazuje, má zvýrazněnou syntaxi. Identifikátory v kódu jsou odkazy, které vedou na stránku obsahující reference na místa v souborech, kde se daný identifikátor vyskytuje (je definován nebo použit). Pomocí těchto odkazů a vyhledávače je zajištěna navigace v kódu.

[1] KDiff3 dostupný z https://kdiff3.sourceforge.net/
[2] WinMerge dostupný z https://winmerge.org/
[3] GitHub – https://github.com/

3.2.3 Vyhodnocení existujících řešení

Existující aplikace pro zobrazování rozdílů v textu jsou pro prezentování výsledků nástroje DiffKemp nedostačující. Zobrazují celé soubory se všemi syntaktickými změnami. Pro prezentování výsledků je sice klíčové zobrazit změny, ale pouze ve funkcích s nalezenými sémantickými rozdíly. Navíc aplikace nesplňují požadavek na zobrazení kontextu (zásobník volání, ...) k nalezeným rozdílům. Nicméně pro vytvoření řešení bude dobré se inspirovat zmíněnými aplikacemi, které zobrazují rozdíly graficky.

Navigace v kódu je sice užitečná, ale z hlediska prezentace daných výsledků si myslím, že je aktuálně nadbytečná. Může se jednat o dobré rozšíření prohlížeče výsledků do budoucna. Důležité je zobrazení kontextu, v jakém je funkce s nalezeným sémantickým rozdílem používaná porovnávanou funkcí.

3.3 Webové technologie

Prohlížeč výsledků nástroje DiffKemp jsem se rozhodl realizovat moderním způsobem – jako webovou aplikaci. Výhodou takového řešení je, že výsledná aplikace je spustitelná téměř na libovolném zařízení a je možné ji popřípadě nasadit na server.

Základem webových aplikací je značkovací jazyk **HTML** (Hypertext Markup Language), který se používá k vytváření obsahu, a jazyk **CSS** (Cascading Style Sheets) sloužící k definování vzhledu stránek. Tato podkapitola se zabývá dalšími webovými technologiemi, které jsou použity v této práci, a to jazykem **JavaScript**, knihovnou **React** a správcem balíčků **npm**.

JavaScript

JavaScript je objektově orientovaný, dynamicky typovaný, interpretovaný jazyk. Obsahuje standardní knihovnu objektů (pole, datum, ...) a je možné ho rozšířit o další objekty sloužící pro různé účely [12].

Používán je jako skriptovací jazyk běžící na straně klienta (v prohlížeči) pro vytváření interaktivních dynamických webových stránek. Pro tento účel poskytuje JavaScript objekty umožňující měnit obsah a vzhled webové stránky a reagovat na uživatelské události.

JavaScript je možné použít také na straně serveru (např. Node.js).

npm

Npm (Node Package Manager) je správce a databáze (tzv. registry) JavaScriptových balíčků. Správce má podobu rozhraní příkazové řádky (CLI), umožňuje vyhledávat a instalovat balíčky z databáze a používat je jako závislosti ve svých projektech [13]. Balíčky jsou ukládány do složky `node_modules` v adresáři projektu.

Npm projekt je popsán souborem `package.json`, který obsahuje jméno, verzi, popis, licenci, seznam závislostí daného projektu, atd. Závislost mapuje jméno balíčku s konkrétním rozsahem verzí.

V případě sdílení projektu se nesdílí obsah složky `node_modules`, ale pouze soubor `package.json` a `package-lock.json`, který obsahuje strom závislostí umožnující, aby následná instalace byla totožná s původní.

React

React je *JavaScriptová* knihovna (framework), která slouží pro vytváření uživatelských rozhraní. Umožňuje vytvářet tzv. SPA (Single Page Application) – jednostránkovou webovou aplikaci, ve které se mění obsah bez znovunačítání stránky. Základem *Reactu* jsou *komponenty* – části uživatelského rozhraní [8], z kterých je tvořena celá webová stránka nebo její část. V závislosti na uživatelských akcí se určuje jaké komponenty budou zobrazeny a jakým způsobem. Pro React existují různé knihovny (již vytvořené komponenty), které je možné ve své aplikaci použít.

React komponenty

Komponenty jsou nezávislé, izolované, znovupoužitelné části uživatelského rozhraní, které spojují obsah (HTML) a logiku (JavaScript) na jednom místě [8]. Mohou být různě velké a často bývají zanořovány do sebe. Komponenty je možné definovat jako JavaScriptové třídy nebo funkce, doporučena je jejich tvorba jako funkce.

Výstupem komponenty je obsah (např. HTML element), který má být zobrazen. Tento výstup bývá v React aplikacích často psán pomocí jazyka JSX.

Komponenty mohou mít také vstupní data – **vlastnosti** (props), které slouží pro komunikaci od rodičovské komponenty k jejímu potomkovi. *Vlastnosti*, které jsou podobné HTML atributům, umožňují potomkům předávat hodnoty, pole, objekty, funkce [8] a tím měnit chování komponenty bez nutnosti znát, jak funguje. Předávání funkcí potomkům pomocí *vlastností* je zejména užitečné pro obsluhu událostí, kdy při výskytu události na zanořené komponentě se zavolá funkce definovaná v nadřazené komponentě.

Pokud je potřeba, aby si komponenta něco pamatovala, tak může mít také vnitřní **stav** (state). Když se *stav komponenty* změní, dochází k *překreslení* (aktualizaci vzhledu komponenty) a to rekurzivním vyhodnocením výstupu komponenty (voláním funkce reprezentující komponentu). Podle výstupu komponenty je upraven DOM[4], podle kterého nakonec prohlížeč „překreslí“ obrazovku [8].

Stav (stavové proměnné) je pro komponentu soukromý a na rozdíl od lokálních proměnných přetrvá i *překreslení* komponenty [8].

React hooky

Pro komponenty psané funkcionálním způsobem jsou důležité tzv. *hooky*. Jedná se o speciální funkce, které zpřístupňují funkcionality Reactu [8].

Jednou z nich je `useState` poskytující stavovou proměnnou pro uchování dat mezi překresleními komponenty. Funkce vrací aktuální hodnotu proměnné a funkci pro její nastavení, při nastavení dochází k zařazení komponenty do fronty komponent, které je potřeba *překreslit*.

JSX

JSX je syntaktické rozšíření JavaScriptu, které umožňuje psát značky podobné XML přímo do JavaScriptu [5]. Pomocí těchto značek je možné psát klasické HTML elementy nebo např. React komponenty.

[4] Document Object Model – objektová reprezentace HTML dokumentu

3.4 Knihovny pro zobrazování rozdílů v textu

Tato sekce se zabývá JavaScriptovými knihovnami pro porovnání textových souborů a zobrazení rozdílů. Cílem je pokusit se najít nejvhodnější knihovnu pro účely této práce.

jsdiff [3]

Knihovna, která slouží pro porovnání textů v JavaScriptu. Umožňuje pro dva zdrojové texty získat např. seznam změn nebo patch se sjednoceným kontextem.

react-diff-viewer [14]

Jedná se o React komponentu pro zobrazování rozdílů v textu. Vstupem jsou dvě verze zdrojových textů, jejichž změny umožňuje vyobrazit v jednom sloupci nebo vedle sebe. Podle počtu stažení je dost používaná, naposledy však byla aktualizovaná před 3 roky. Je poměrně jednoduchá na používání a umožňuje:

- doplnit text o **zvýraznění syntaxe** (např. s využitím knihovny PrismJS),
- **změnit výchozí vzhled**,
- **zobrazit celý text nebo pouze rozdíly** s možností zobrazit skryté (nezměněné) části kliknutím na tlačítko,
- vybrat metodu porovnání (po slovech, znacích, řádcích, . . .).

react-diff-view [20]

Tento balíček respektive React komponenta umožňuje stejně jako ta předchozí zobrazit rozdíly v textu vedle sebe nebo v jednom sloupci. Komponenta jako vstup potřebuje tentokrát textový rozdíl se sjednoceným kontextem zdrojových textů. Oproti předchozímu balíčku je méně populární (podle počtu stažení), ale je pravidelně udržována. Používání této knihovny je více složité, v základu zobrazuje pouze změněné části textu. Balíček umožňuje:

- **upravit zobrazovaný text** – poskytuje řadu nástrojů, které umožňují např.: filtrovat změny nebo rozšířit zobrazovaný text o nezměněné části ze zdrojového textu,
- zobrazit vlastní komponenty kolem bloků s rozdíly – lze využít např. na vytvoření tlačítka pro zobrazení skrytých (nezměněných) části textu,
- **zvýraznit syntax** v textu pomocí systému tokenů (s využitím refractor knihovny),
- **upravit výchozí vzhled** (pomocí CSS stylů),
- přidat vlastní rozšíření pomocí tzv. widgetů – např. přidat možnost komentování změn.

Vyhodnocení knihoven

Jako nejvhodnější knihovna pro zobrazování rozdílů v kódu pro účely této práce byla vybrána knihovna **react-diff-view**. Důvodem je především to, že knihovna poskytuje větší možnosti toho, co se bude zobrazovat (rozšířit a filtrovat zobrazovaný text). Tato vlastnost se bude pravděpodobně hodit pro vybrání potřebných částí k zobrazení při prezentování výsledků nástroje DiffKemp.

Kapitola 4

Prohlížeč výsledků

Jak již bylo zmíněno v Sekci 2.4, tak původní prezentace výsledků (nalezených rozdílů) nástroje DiffKemp má pouze textovou podobu ukládanou do souborů nebo vypsanou na standardní výstup, která není úplně uživatelsky přívětivá. Jelikož existující aplikace sloužící k zobrazování rozdílů (jako např. Meld) jsou nedostačující pro prezentování výsledků nástroje DiffKemp (jak bylo popsána v Sekci 3.2), zabývá se tato kapitola návrhem nového způsobu prezentace výsledků, jehož cílem je být více uživatelsky přívětivý než původní způsob. Navržené řešení by mělo splňovat požadavky uvedené v Sekci 3.1.

Pro příjemnější procházení je zvoleno prezentaci výsledků realizovat jako aplikaci (prohlížeč výsledků) místo prezentace ve formě textových souborů. V původním řešení musel uživatel pro zobrazení nalezených rozdílů konkrétní porovnávané funkce najít a otevřít soubor s názvem dané funkce. Obdobně by mohl uživatel i v novém prohlížeči nejprve vybrat porovnávanou funkci, pro niž by si chtěl nechat zobrazit rozdíly.

Původní soubor s nalezenými rozdíly pro porovnávanou funkci obsahuje za sebou vypsány jednotlivé nalezené rozdíly, které obsahují název funkce, ve které byl nalezen rozdíl, zásobníky volání a textový rozdíl sémanticky různé funkce. V tomto případě je nalezení konkrétní rozdílné funkce v původním řešení poměrně nepraktické, protože je potřeba projít soubor, ve kterém jsou uvedeny zásobníky volání a textové rozdíly funkcí, které uživatele nemusí zrovna zajímat. Proto nové řešení bude po výběru porovnávané funkce zobrazovat pouze názvy rozdílných funkcí (pro jednoduší nalezení rozdílné funkce uživatelem) a více informací o rozdílu zobrazí až po výběru konkrétní funkce.

Stejně jako v původním řešení chceme, aby pro nalezený rozdíl byl zobrazen zásobník volání a syntaktický rozdíl sémanticky různé funkce. Navíc chceme uživateli usnadnit analýzu rozdílu a to poskytnutím kontextu ve formě zdrojových kódů pro jednotlivé funkce v zásobníku volání. Ve zdrojových kódech chceme vyznačit řádek s následující volanou funkcí ze zásobníku a v případě, že se jedná o zdrojový kód rozdílné funkce chceme v něm vyznačit syntaktické rozdíly.

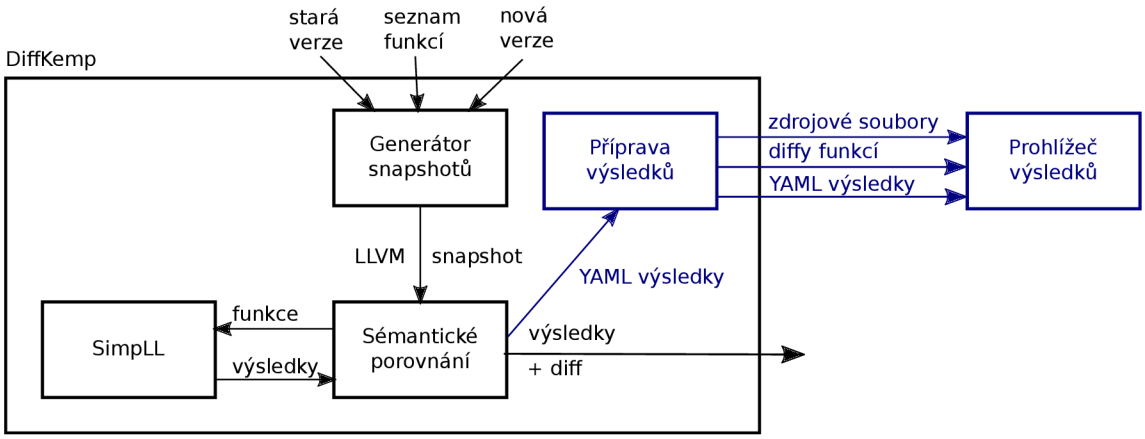
Prohlížeč jsem se rozhodl realizovat jako webovou aplikaci s využitím technologií popsaných v Sekci 3.3. K samotnému zobrazování rozdílů ve funkcích byla v Sekci 3.4 vybrána knihovna react-diff-view. Podrobnějším návrhem uživatelského rozhraní a způsobem prohlížení výsledků se zabývá Sekce 4.2, jejímž cílem je, aby navržené řešení splňovalo specifikované požadavky.

K zobrazení nalezených rozdílů bude potřebovat prohlížeč výsledků získat nějakým způsobem informace o těchto rozdílech, proto se Sekce 4.3 zabývá návrhem nového výstupu sémantického porovnání, který bude sloužit jako rozhraní nástroje DiffKemp k propojení

s prohlížečem výsledků. Tento výstup bude poskytovat potřebné informace k vizualizaci v serializované podobě za účelem snadného zpracování.

Podrobnějším návrhem rozšíření architektury DiffKemp o nový způsob prezentace výsledků se zabývá následující Sekce 4.1.

4.1 Návrh architektury



Obrázek 4.1: Architektura nástroje DiffKemp doplněná o další části sloužící k novému způsobu prezentování výsledků

Na Obrázku 4.1 je možné vidět doplnění původní architektury nástroje DiffKemp (popsané v Sekci 2.1) o nové části – **nový výstup** sémantického porovnání (YAML výsledky), samotný **prohlížeč výsledků** a **přípravu výsledků** k zobrazování. Jednotlivé části budou více vysvětleny v následujícím textu.

Jelikož prohlížení nalezených rozdílů nemusí být úplně typickou operací nástroje DiffKemp, bude prohlížeč **samostatnou částí nástroje**, kterou bude moci uživatel využít po provedené sémantické analýze programů k přehlednému zobrazení nalezených rozdílů. Ke spuštění prohlížeče bude sloužit nový příkaz nástroje DiffKemp.

Protože prohlížení výsledků bude samostatnou částí nástroje, bude potřebovat prohlížeč nějak získat informace o výsledcích analýzy. Původní podoba výsledků (popsaná v Sekci 2.4) není úplně nejlepší pro strojové zpracování a navíc neobsahuje všechny potřebné informace pro vizualizaci (např. údaje pro zobrazení zdrojových kódů funkcí), proto je sémantické porovnání rozšířeno o **nový výstup** (YAML výsledky), který bude obsahovat serializované výsledky a další data potřebná pro vizualizaci nalezených rozdílů prohlížečem. Návrh formátu tohoto výstupu je popsán v Sekci 4.3.

V prohlížeči budeme chtít zobrazovat syntaktické rozdíly funkcí s nalezeným sémantickým rozdílem, k tomu byla v Sekci 3.4 vybrána knihovna react-diff-view, která potřebuje mít na vstupu **textový rozdíl** souborů se *sjednoceným kontextem*. Protože rozdíl (diff) poskytovaný jako původní výstup sémantického porovnání má *kopírovaný kontext* (rozdíl mezi kopírovaným a sjednoceným kontextem je zmíněn v Sekci 3.2) a je ukládán společně se zásobníky volání a všemi ostatními rozdílnými funkcemi, nebude úplně vhodný k použití. Z toho důvodu bude nutné někde rozdíl se sjednoceným kontextem vytvořit. Možností by bylo vytvořit ho až v samotném prohlížeči ze zdrojových kódů a to např. knihovnou jsdiff

zmíněnou Sekci 3.4. Proto, aby prohlížeč nebyl zbytečně zatěžován tímto procesem, bude vhodné vytvořit tyto rozdíly před samotným spuštěním prohlížeče.

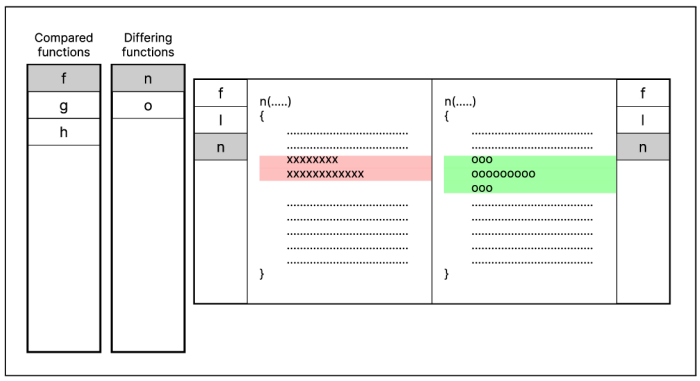
Kromě rozdílů budeme chtít v prohlížeči zobrazovat také zdrojové kódy funkcí ze zásobníku volání. Jelikož zdrojové kódy volajících funkcí mohou být teoreticky také syntakticky různé (aniž by byly sémanticky různé), bude vhodné k jejich zobrazování použít stejnou knihovnu (react-diff-view). Jelikož knihovna potřebuje textový rozdíl se sjednoceným kontextem, bude potřeba vytvořit textové rozdíly pro všechny funkce ze zásobníku volání

Textové rozdíly je možné teoreticky provést po jednotlivých souborech, ale vzhledem k tomu, že funkce může být mezi verzemi programu např. přesunuta do jiného souboru, bude vhodné ho tvořit po samotných funkcích. K tomu bude nutné vědět informace o jejich definicích – (1) v jakém souboru se funkce nachází, (2) řádek kde začíná a (3) řádek kde končí.

Pro prohlížeč výsledků bude potřeba předpřipravit již zmíněné textové rozdíly funkcí, ke kterým bude nutné umožnit prohlížeči přístup. Kromě toho bude muset mít prohlížeč přístup k výsledkům analýzy (YAML výsledky) a pro zobrazení zdrojových kódů funkcí bude muset mít také přístup k starým zdrojovým souborům, ve kterých se funkce nacházejí. Tyto akce bude mít na starosti nová část architektury – fáze **přípravy výsledků**, která bude využívat nalezených rozdílu ze sémantického porovnání, přesněji řečeno nově vytvořeného výstupu této fáze. Přístup prohlížeče k jednotlivým částem (souborům) bude zajištěn statickým serverem, který vrátí potřebné soubory.

4.2 Návrh uživatelského rozhraní

Tato kapitola se zabývá návrhem nového způsobu vizualizace výsledků nástroje DiffKemp. Cílem je vytvořit uživatelské rozhraní prohlížeče výsledků, které bude umožňovat snadnou navigaci v nalezených rozdílech a jednotlivé rozdíly bude přehledně zobrazovat. Při tvorbě návrhu bude využito specifikace požadavků z Tabulky 3.1.



Obrázek 4.2: Prvotní návrh prohlížeče výsledků

Na Obrázku 4.2 je možné vidět prvotní návrh prohlížeče výsledků. Obrázek zachycuje zobrazení výsledku porovnané funkce `f` s nalezeným rozdílem ve funkci `n`, která byla volána z funkce `f` funkcí `l`.

K výběru nalezeného rozdílu, který se má zobrazit, slouží levá část návrhu. V té se nachází dva sloupce sloužící pro výběr funkcí. První sloupec obsahuje názvy *porovnaných* funkcí, druhý sloupec slouží k výpisu jmen funkcí, u kterých nástroj DiffKemp odhalil sémantické odlišnosti pro zvolenou porovnávanou funkci.

Pravá část návrhu je určena k vyobrazení *nalezeného rozdílu* pro zvolenou porovnávanou a rozdílnou funkci. Informace o nalezeném rozdílu, které nástroj DiffKemp poskytuje, se skládají ze zásobníků volání a textového rozdílů funkce s nalezeným sémantickým rozdílem. Střed pravé části návrhu je vymezený pro zobrazení textového rozdílu, pro poskytnutí lepšího kontextu uživateli ho chceme zobrazit a vyznačit ve zdrojovém kódu funkce. Na bocích této části se vyskytují *zásobníky volání* – vlevo zásobník pro starou verzi programu, vpravo pro novou. Zásobníky v původním výstupu obsahovaly jména volaných funkcí, soubory, ve kterých byly volány, a čísla řádků, na kterých byly volány. Zásobníky v prohlížeči budou zobrazovat pouze jména volaných funkcí, pro větší přehlednost budou samotná volání vyznačena ve zdrojovém kódu funkcí místo toho, aby byla vypsána jen jako text. Kromě názvů volaných funkcí bude zásobník obsahovat na prvním místě název porovnávané funkce. Jednotlivé názvy funkcí v zásobnících budou sloužit k zobrazení jejich zdrojového kódu, který se bude vykreslovat ve středové části. Podle toho, jestli se bude jednat o volající nebo rozdílnou funkci, bude buďto vyznačen řádek s volanou funkcí nebo syntaktický rozdíl. Při prvním zobrazení (po výběru porovnávané a rozdílné funkce) bude nejprve zobrazen zdrojový kód rozdílné funkce, jelikož se jedná o tu nejdůležitější informaci.

Následuje úsek textu, který se zaměřuje se na jednotlivé části návrhu více dopodrobna a navíc se ho snaží upravit tak, aby výsledný návrh byl co nejlepší.

Zobrazení nalezeného rozdílu ve funkci

Syntaktický rozdíl v rozdílné funkci by bylo možné zobrazit v jednom sloupci (sjednoceně), ale řekl bych, že přehlednější je zobrazit jednotlivé rozdíly (funkce) vedle sebe – vlevo starou verzi funkce, vpravo novou.

Podle požadavku č. 4 z Tabulky specifikace požadavků 3.1 má být rozdíl zobrazen ve **zdrojovém kódu funkce**. Vzhledem k tomu, že kód funkce může být v některých případech rozsáhlý (zabírající mnoho řádků), bylo by vhodné, kdyby se v základu zobrazovaly pouze rozdíly (v dané funkci) s kontextem jednoho řádku, aby se rozdíly „neztratily“ v množství kódu. Pro ohraničení rozdílu by chtělo také zobrazit začátek a konec funkce (první a poslední řádek funkce). Zbylé části funkce by se mohly zobrazit až po kliknutí na tlačítko, které by se zobrazovalo v místě „chybějících“ řádků kódu. Obdobně tomu tak je u služby GitHub při zobrazování rozdílů v souborech mezi jednotlivými revizemi.

Podle požadavku č. 9 mají být zobrazeny čísla jednotlivých řádků kódu. Také má být zobrazena cesta k souboru s danou funkcí (požadavek č. 10) – ta by mohla obsahovat název snapshotu a relativní cestu vůči němu. Vhodné by rovněž bylo, aby prohlížeč dokázal zobrazit alespoň **80 znaků kódu na jeden řádek**.

Zobrazení kódu pro volající funkce

Kromě zobrazení samotného rozdílu má podle požadavku č. 6 z Tabulky 3.1 umět prohlížeč zobrazit zdrojový kód pro jednotlivé funkce ze zásobníku volání. Ten by teoreticky mohl být zobrazen pouze jednou, avšak kód funkce se mohl změnit (bez změny sémantiky), proto bude vhodné zobrazovat i kód těchto funkcí vedle sebe. V kódu má být také zvýrazněn řádek volané funkce (požadavek č. 8).

Jak již bylo zmíněno v předešlé části, bude vhodné zmenšit množství zobrazovaného kódu – v tomto případě zobrazováním celé funkce pokud nebude příliš rozsáhlá (např. do 25 řádků), jinak zobrazováním pouze začátku a konce funkce a kontextu např. 10 řádků kolem zvýrazněného řádku s voláním. Zbylé části funkce bude možné zobrazit až po kliknutí na tlačítko.

Zobrazení zásobníků volání

Podle požadavku č. 5 má být zobrazen zásobník volání. V prvotním návrhu se počítalo se zobrazením obou zásobníků volání. Vzhledem k tomu, že chceme zobrazit alespoň 80 znaků kódu na jeden řádek a zobrazit takto kódy dvou funkcí vedle sebe, bude pravděpodobně nemožné nebo obtížné zobrazit oba zásobníky na obrazovkách s nižším rozlišením. Navíc podle analýzy rozdílů ve vybraných verzích jádra RHEL[1] uvedené v Tabulce 4.1 vyplynulo, že ve většině případů jsou zásobníky (s uvedenými jmény funkcí) totožné. Bylo by tudíž zbytečné zobrazovat dva stejné zásobníky, proto bude lepší zobrazovat zásobník pouze jeden.

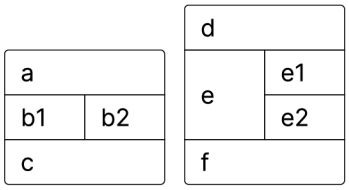
Tabulka 4.1: Tabulka zachycující analýzu totožnosti (podle jmen funkcí) zásobníků nalezených rozdílů nástrojem DiffKemp pro různé verze jádra RHEL (hlavní zaměření nástroje)

Verze jádra	**Počet nalezených rozdílů**	**Počet totožných zásobníků (%)**	**Počet rozdílných zásobníků (%)**
8.0/8.1	207	204 (98,55)	3 (1,45)
8.1/8.2	427	424 (99,30)	3 (0,70)
8.2/8.3	414	380 (91,79)	34 (8,21)

Problém nastává v případě, kdy jsou **zásobníky různé**. To v analyzovaných zásobnících nastává např. kvůli tomu, že v jedné verzi zásobníku:

- je volána jiná funkce (makro) než ve druhé,
- je voláno více funkcí (maker) než ve druhé.

Možné řešení by bylo v tomto případě zobrazit oba zásobníky, obdobně jak tomu bylo v prvotním návrhu. Z důvodu konzistence uživatelského rozhraní a z důvodu redukce zobrazení duplicitních informací, je navrženo jiné řešení – **zobrazení pouze jednoho zásobníku a jeho rozdělení na dvě části** v případě různých volaných funkcí. Levá část by obsahovala volání ve staré verzi, pravá v nové. Toto řešení umožní navíc uživateli snadněji rozpoznat změny v zásobnících. Návrh zásobníku je zachycen na Obrázku 4.3.



Obrázek 4.3: Návrh vzhledu zásobníku volání v případě, že jednotlivé funkce budou různé. Vlevo je případ, kdy dochází k volání jiné funkce (`b2` místo `b1`), Vpravo je zachyceno zajímavé rozvětvení v případě, že místo funkce `e` jsou volány funkce `e1` a `e2`.

Navigace – výběr porovnávané a rozdílné funkce

Jelikož pro sloupce výběru porovnávané a rozdílné funkce by zůstalo na stránce (obrazovce) nedostatek místa, je navrženo nové řešení – samostatná stránku pro výběr porovnané funkce, následovaná stránkou pro výběr rozdílné funkce, po jejímž vybrání se zobrazí stránka vizualizující nalezený rozdíl.

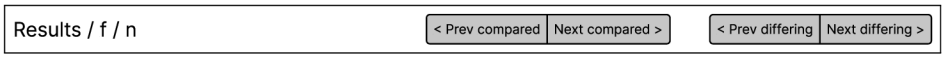
[1] Red Hat Enterprise Linux – hlavní zaměření nástroje DiffKemp

Zobrazený nalezený rozdíl je dán dvojicí (porovnaná funkce, rozdílná funkce). Porovnaná funkce se může lišit ve více funkcích. Stejně tak funkce, ve které je nalezen rozdíl, může být rozdílná pro více porovnaných funkcí (Sekce 2.3). Budeme chtít, aby uživatel mohl při zobrazování nalezeného rozdílu snadno zobrazit i další funkce spojené s daným rozdílem, jedná se o funkce:

- porovnávané – proto, aby mohl zjistit jaké porovnávané funkce (např. tvořící rozhraní knihovny) jsou ovlivněny daným nalezeným rozdílem,
- rozdílné – pro zjištění v jakých dalších funkcích se porovnávaná funkce liší.

Toto se snaží vyřešit **navigace** zachycená na Obrázku 4.4, která obsahuje informaci o tom, jaký rozdíl je zobrazen – `porovnaná funkce` (f) `/ rozdílná funkce` (n). Po kliknutí na název porovnané funkce se zobrazí seznam jmen rozdílných funkcí pro danou porovnávanou funkci. Při kliknutí na jméno rozdílné funkce se zobrazí seznam jmen porovnaných funkcí, které se liší v dané rozdílné funkci. `Results` slouží k zobrazení všech porovnaných funkci. Dále navigace obsahuje šipky pro přepínání na:

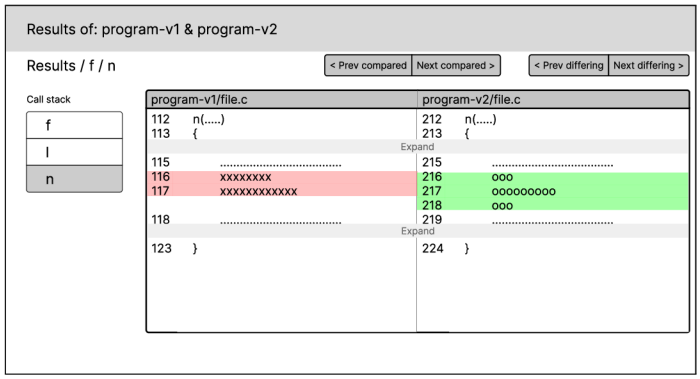
- první rozdílnou funkci předcházející/další porovnané funkce,
- předcházející/další rozdíl aktuální porovnávané funkce.



Obrázek 4.4: Návrh navigace pro zobrazený rozdíl

Výsledný návrh

Na Obrázku 4.5 je možné vidět hlavní část (zobrazení nalezeného rozdílu) výsledného návrhu prohlížeče.



Obrázek 4.5: Výsledný návrh zobrazení nalezeného rozdílu. Oproti prvotnímu návrhu má (1) jiný způsob navigace ve výsledcích, (2) zobrazuje pouze jeden zásobník místo dvou, (3) zobrazuje čísla řádků ve zdrojovém kódu, (4) vypisuje umístění funkce a (5) informuje o porovnaných verzích (složkách) programu.

4.3 Návrh formátu nového výstupu nástroje DiffKemp

Tato sekce se zabývá návrhem nového výstupu fáze sémantického porovnání. Cílem výstupního formátu je poskytnout prohlížeči snadno zpracovatelné informace o nalezených rozdílech a dalších údajích potřebných pro vizualizaci. K serializaci dat byl vybrán formát YAML[2].

```
old-snapshot: /home/lukas/diffkemp/snapshot/linux-8.0
new-snapshot: /home/lukas/diffkemp/snapshot/linux-8.1
results:
- function: __alloc_workqueue_key
  diffs:
  - function: worker
    old-callstack:
    - name: init_rescuer
      line: 4094
      file: kernel/workqueue.c
    new-callstack:
    - ...
definitions:
  init_rescuer:
    kind: function
    old:
      file: kernel/workqueue.c
      line: 4005
      end-line: 4030
    new:
      ...
```

Výpis 4.1: Zkrácený příklad nového výstupu fáze sémantického porovnání ve formátu YAML

Na Výpisu 4.1 je možné vidět ukázku navrženého výstupního formátu. Následuje popis jednotlivých položek s uvedením jejich účelu:

- `old-snapshot`, `new-snapshot` – absolutní cesty k adresářům s vytvořenými snapshoty jednotlivých verzí programu. V těchto adresářích jsou uloženy *zdrojové soubory*, které budou nutné pro vizualizaci kódu a vytváření textových rozdílů funkcí.
- `results` – nalezené rozdíly pro porovnané funkce (položka napodobuje původní výstup sémantického porovnání). Jedná se o seznam porovnaných funkcí, pro které byl nalezen sémantický rozdíl. Jednotlivé položky obsahují:
  - název porovnané funkce (`function`),
  - seznam nalezených rozdílů (`diffs`). Součástí rozdílu je název rozdílné funkce (`function`) a zásobníky volání pro starou (`old-callstack`) verzi programu a novou (`new-callstack`). Zásobníky obsahují volání – jméno volané funkce (`name`), řádek, na kterém je funkce volána (`line`) a soubor (`file`), ve kterém je volána.

[2]YAML – YAML Ain't Markup Language https://yaml.org/

Díky této položce bude prohlížeč schopný (1) umožnit navigaci ve výsledcích (výběr z porovnaných a rozdílných funkcí), (2) zobrazit zásobníky volání a (3) zvýraznit řádek v kódu s volanou funkcí.

- `definitions` – definice funkcí vyskytujících se v položce `results`. Jedná se o slovník, kde klíčem jsou názvy funkcí a hodnotou informace o definicích. Slovník je to z důvodu snadnějšího dohledávání. Jednotlivé definice obsahují informace pro starou (`old`) a novou (`new`) verzi programu o tom, v jakém souboru se funkce nachází (`file`), kde začíná (`line`) a kde končí (`end-line`). Kromě toho je zde také položka `kind` udávající, jestli se jedná o funkci, datový typ nebo makro. V případě, že nová verze má odlišné jméno (`name`), bude toto jméno uloženo v položce `new`.

  Účelem této položky je poskytnutí informací pro tvorbu textových rozdílů po funkcích při fázi *přípravy výsledků*. Při této tvorbě budou definice jednotlivých funkcí doplněny o položku `diff`, která bude nabývat hodnoty `true` v případě, že pro funkci bude existovat syntaktický rozdíl. Dále se tato data budou hodit prohlížeči k zobrazení kódu funkcí.

Navržený výstup může sloužit nejen prohlížeči pro zobrazování nalezených rozdílů, ale také pro jiné účely. Může být použit např. k analýze výsledků. Pomocí tohoto výstupu byla vytvořena analýza uvedená v Tabulce 4.1.

Kapitola 5

Implementace rozšíření

Tato kapitola se zaměřuje na implementaci rozšíření nástroje DiffKemp o prohlížeč výsledků. První Sekce 5.1 se zabývá vytvořením nového výstupu fáze sémantického porovnání, který obsahuje výsledek analýzy a další potřebné informace pro jeho zobrazení. Další Sekce 5.2 realizuje přípravu výsledků, ke které je využit daný výstup, a spuštění samotného prohlížeče. Poslední Sekce 5.3 se věnuje implementaci prohlížeče výsledků jako webové aplikace.

5.1 Nový výstup fáze sémantického porovnání

Vytváření nového výstupu fáze sémantického porovnání, jehož formát je popsán v Sekci 4.3, je implementováno v Python části nástroje DiffKemp ve třídě `YamlOutput`. Pro vytvoření souboru ve formátu YAML je použit balíček PyYaml[1]. Tento výstup je dále využit pro přípravu a zobrazování výsledků.

Pokud není při sémantickém porovnání zvolen výpis výsledků na standardní výstup, tak dochází k vytvoření tohoto výstupu (souboru ve formátu YAML). Ten je uložen společně s původními výstupy (prezentací) sémantického porovnání do uživatelem vybraného adresáře. Nový výstup je uložen do souboru s názvem `diffkemp-out.yaml`.

Informace pro vytvoření výstupu jsou získány ze sestaveného výsledku a z grafu volání (popsaných v Sekci 2.3.3), který obsahuje informace o *definicích funkcí*. Kromě rozdílů ve funkcích je DiffKemp schopný nalézt rozdíly také v makrech nebo datových typech. Získání definic datových typů je obtížnější, jelikož jsou uloženy mimo graf, ze kterého na ně existují odkazy v uzlech grafu (funkcích), které tyto datové typy používají. Makra nejsou v grafu také uložena, volání začínající voláním makra jsou uložena zvlášť, pro tato volání chybí informace o definicích.

Původní sestavený výsledek v programu (Python části) obsahoval zásobníky volání v již textové formě, ze kterých by bylo možné získat jednotlivé údaje např. regulárním výrazem. Z hlediska budoucího vývoje nástroje to není úplně nejlepší řešení, proto došlo k refaktorizaci zásobníku na reprezentaci pomocí nové třídy `Callstack`, ve které jsou jednotlivé údaje lépe přístupné.

[1] PyYAML – https://pypi.org/project/PyYAML/

5.2 Fáze přípravy výsledků

Příprava výsledků je realizována jako nový příkaz (`view`) nástroje DiffKemp, který používá výsledek sémantického porovnání ve formátu YAML (dále jen „výsledek“). Tento příkaz dělá následující:

- Pro funkce vyskytující se v definicích výsledku **vytváří textový rozdíl** se sjednoceným kontextem. K tomu je potřebné získat zdrojové soubory, ve kterých jsou funkce umístěny. Cesty k adresářům se zdrojovými soubory jsou uloženy ve výsledku jako položky `old-snapshot` a `new-snapshot`, v samotných definicích jsou pak uvedeny relativní cesty ke zdrojovým souborům v těchto adresářích. V definicích jsou také uloženy informace o začátcích a koncích funkcí, které jsou též potřebné k vytváření rozdílů.

  Nástroj DiffKemp již obsahoval funkci pro vytváření textových rozdílů s kopírovaným kontextem, došlo k refaktorizaci této funkce, aby mohla být použita i pro vytváření nového rozdílu. Jelikož se rozdíl tvoří jenom pro úsek kódu obsahující funkci, je potom nutné upravit informace o tom, kde změny začínají tak, aby čísla odpovídala umístění funkce v souboru. V případě sjednoceného kontextu je před každým místem změn (tzv. hunk) řádek ve formátu `@@ -začátek,počet +začátek,počet @@`, kde vlevo je informace o staré verzi, vpravo o nové [2]. Tyto začátky (číslo řádku, kde změna začíná) je nutné upravit podle čísla řádku, na kterém začíná funkce.

  Rozdíl mezi kopírovaným a sjednoceným kontextem je zmíněn v Sekci 3.2.

  V načteném výsledků dochází k doplnění definic funkcí o údaj, zda je funkce syntakticky rozdílná či nikoliv.

- Dále jsou (1) staré zdrojové soubory, (2) vytvořené rozdíly funkcí a (3) upravený výsledek uloženy do adresáře, ze kterého tyto soubory může poté získat prohlížeč pomocí asynchronní komunikace.

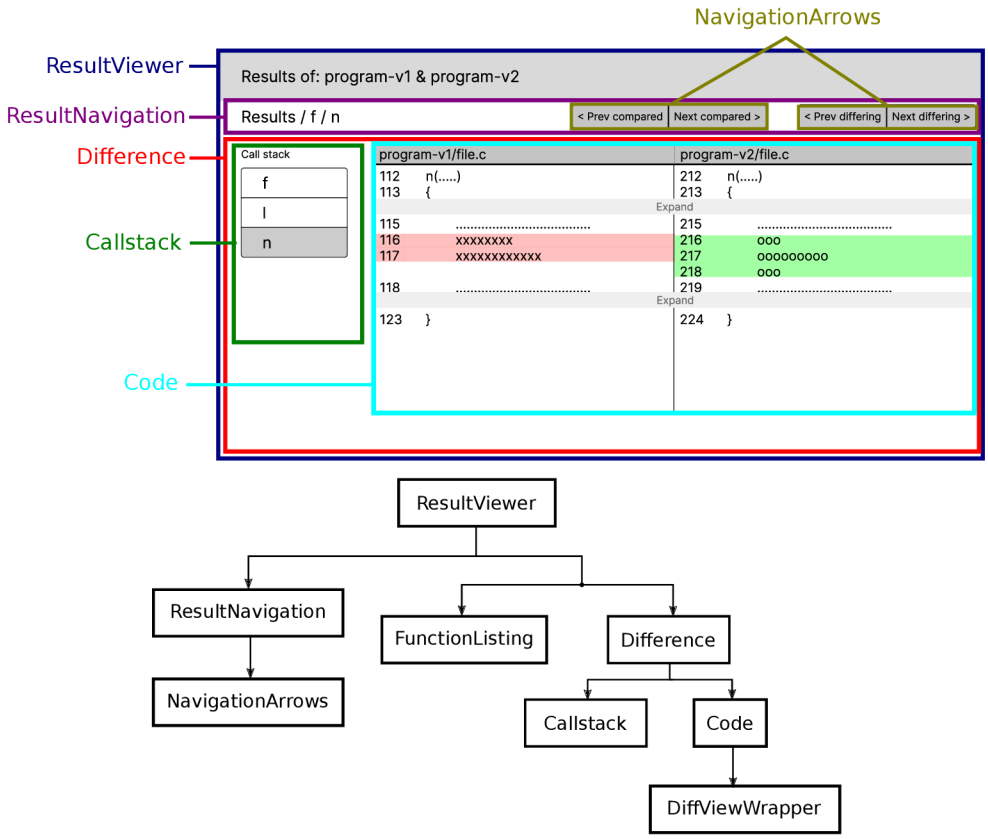
- Nakonec příkaz spustí server, na kterém běží prohlížeč výsledků. Standardně je spuštěn statický server serve[2], který spouští optimalizované a minimalizované sestavení aplikace (prohlížeče výsledků).

  Příkaz pro přípravu výsledku je také možné spustit s přepínačem `--devel`, který spustí aplikaci pro účely vývoje umožňující např. ladění.

5.3 Implementace prohlížeče výsledků

Prohlížeč výsledků byl realizován pomocí JavaScriptové knihovny React, pro vývoj byl využit nástroj Create React App[3], který umožňuje např. spuštění aplikace ve vývojářském režimu. Použit byl také frontendový framework Bootstrap[4] respektive jeho Reactová alternativa React Bootstrap[5], která poskytuje řadu užitečných komponent kromě jiného např. pro rozložení obsahu na stránce.

[2] serve – https://www.npmjs.com/package/serve
[3] Create React App – https://create-react-app.dev/
[4] Bootstrap – https://getbootstrap.com/
[5] React Bootstrap – https://react-bootstrap.github.io/



Obrázek 5.1: Rozložení výsledného návrhu uživatelského rozhraní zobrazující nalezený rozdíl na jednotlivé komponenty a zachycení jejich hierarchie

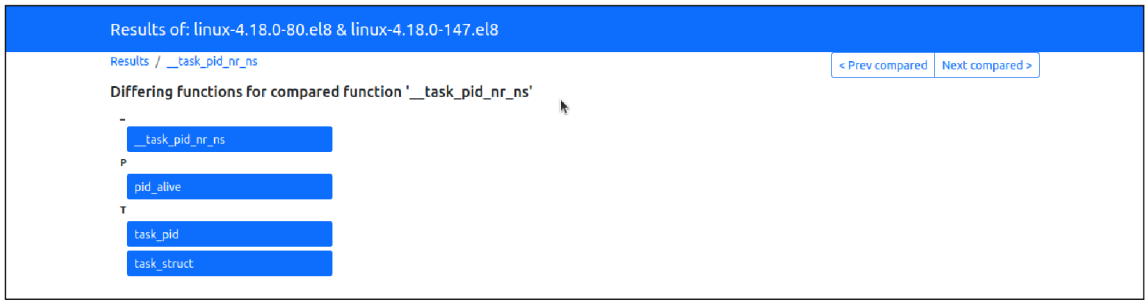
Na Obrázku 5.1 je možné vidět rozložení výsledného návrhu hlavní části prohlížeče (zobrazující nalezený rozdíl) na jednotlivé komponenty a ve spodní části hierarchii jednotlivých komponent.

Hlavní komponentou je `ResultViewer`, který nejprve načte soubor ve formátu YAML popisující výsledek sémantického porovnání (dále jen „výsledek“) a uloží ho jako instanci třídy `Result`. Získání souborů je zajištěno funkcí `fetch`, která vrací obsah daného souboru. Pro zpracování formátu YAML je použit balíček JS-YAML[6]. Třída `Result` kromě samotného výsledku poskytuje metody pro navigaci (např. získání další rozdílné funkce pro porovnávanou) a pro získání seznamu funkcí (např. porovnaných funkcí, které se liší v určité rozdílné funkci).

Komponenta `ResultViewer` slouží k uchování aktuálně vybrané porovnávané a rozdílné funkce. Vykresluje komponentu `ResultNavigation`, která společně s `NavigationArrows` slouží k informování uživatele o vybraných funkcích a umožňuje mu navigaci ve výsledcích.

Po spuštění prohlížeče se vypíše seznam všech porovnaných funkcí, ve kterých byl nalezen rozdíl. To společně s výběrem funkce má na starosti komponenta `FunctionListing`, která slouží nejen k tomu, ale i k výpisu dalších funkcí (rozdílných pro porovnávanou, porovnaných pro rozdílnou). K získání seznamu funkcí využívá metod třídy `Result`. Tímto jsou splněny požadavky 1 a 2 ze specifikace požadavků 3.1. Na Obrázku 5.2 je snímek, který zachycuje výpis funkcí ve vytvořeném prohlížeči.

[6] JS-YAML – `https://www.npmjs.com/package/js-yaml`



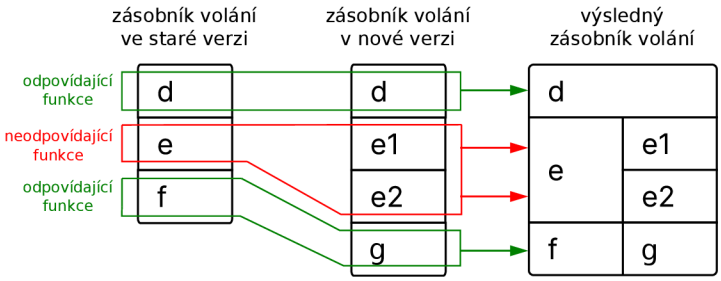
Obrázek 5.2: Snímek vytvořeného řešení pro výpis funkcí. Na snímku je zachycen výpis rozdílných funkcí pro porovnávanou funkci (`__task_pid_nr_ns`), obdobně vypadá i výpis názvů všech porovnaných funkcí.

Pokud je vybrána jak porovnávaná, tak rozdílná funkce, místo `FunctionListing` se vykreslí komponenta `Difference`, která slouží k zobrazení nalezeného rozdílu (požadavek č. 3). Komponenta zobrazuje zásobník volání (`Callstack`) a kód funkce (`Code`). Pomocí navigace je poté možné se vrátit k zobrazení:

- seznamu rozdílných funkcí pro vybranou porovnávanou,
- seznamu porovnaných funkcí, které se liší ve vybrané rozdílné,
- seznamu všech porovnaných funkcí, ve kterých byl nalezen rozdíl,
- předcházejícího/následujícího rozdílu pro stejnou porovnávanou funkci,
- rozdílu pro předcházející/následující porovnávanou funkci,

Zásobník volání

Zásobník volání (`Callstack`) slouží k výpisu názvů funkcí a obsahuje porovnávanou funkci a volané funkce vedoucí k funkci s nalezeným sémantickým rozdílem, která je vypsána jako poslední. Výsledný zobrazený zásobník je tvořen spojením zásobníků volání ve staré a nové verzi, které byly načteny ze souboru popisujícího výsledek (YAML).



Obrázek 5.3: Obrázek se snaží zachytit proces vytváření výsledného zásobníku volání podle zásobníků ve staré a nové verzi, ve kterých se hledají páry odpovídajících si funkcí.

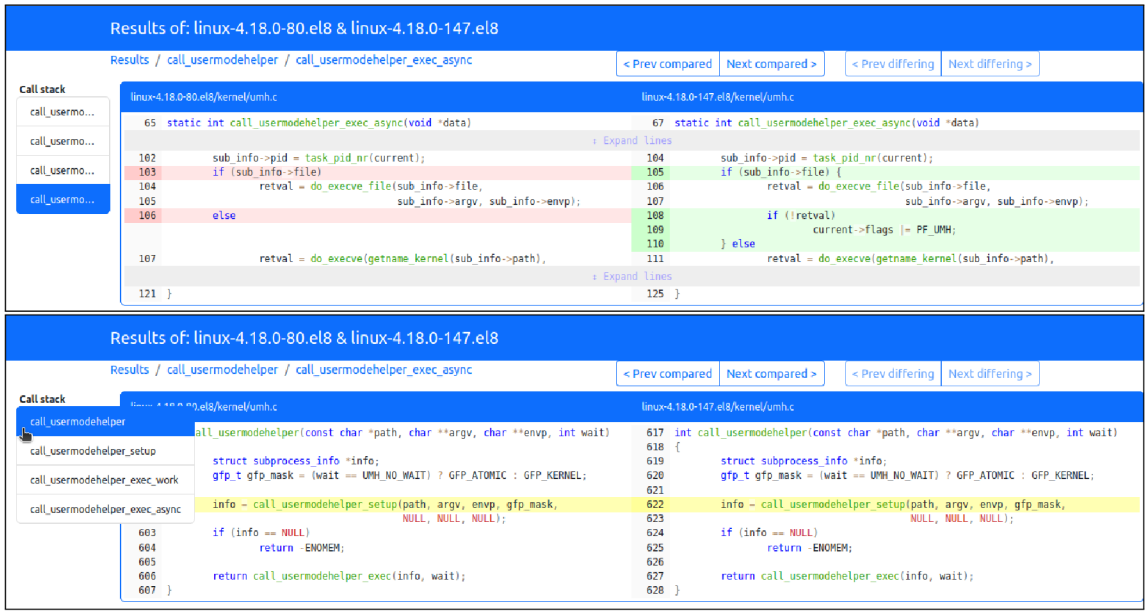
Zobrazení zásobníku probíhá tak, že se v zásobnících hledají páry odpovídajících si funkcí, které se zobrazují na stejném řádku. Za odpovídající funkce jsou považovány ty, které (1) se nachází na konci zásobníku (funkce obsahující rozdíl) a ty, které (2) mají stejné

jméno. Pokud mají funkce z páru úplně stejný název, tak je zobrazen tento název jenom jednou, jinak je vedle sebe zobrazen název ze starého zásobníku a vedle z nového zásobníku. Funkce, pro které nebylo možné najít odpovídající funkci, jsou zobrazeny tak, že nalevo jsou zobrazeny pod sebou jednotlivé názvy funkcí starého zásobníku, napravo z nového zásobníku. Obrázek 5.3 se snaží zachytit proces tvorby výsledného zásobníku. K zobrazení jednotlivých funkcí slouží další komponenty, které využívají Bootstrap komponent `ListGroup` pro zobrazení skupiny názvů a `ListGroup.Item` pro zobrazení samotného názvu. Názvy maker a datových typů jsou zobrazeny tak, že pod názvem je v závorce uvedeno `type` pro datové struktury a `macro` pro makra (požadavek č. 11).

Z vypsaných funkcí si může uživatel vybrat funkci, jejíž kód si chce nechat zobrazit, implicitně je vybrána rozdílná funkce. Aktuálně zobrazovaná funkce je v zásobníku zvýrazněna.

Šířka zásobníku je dána podle toho, aby v kódu bylo možné zobrazit alespoň 80 znaků na jednom řádku. Dosaženo je toho s využitím CSS pravidla `@media`, s jehož pomocí je pro obrazovky se šířkou rozlišení 1280px, 1366px a 1920px upravena šířka zásobníku tak, aby bylo možné zobrazit požadovaný počet znaků v kódu. Jedná se o rozlišení, které jsou běžně používané pro obrazovky. Pro rozlišení obrazovky s menší šířkou je zásobník volání umístěn nad zobrazeným kódem.

Pokud je zásobník nedostatečně široký proto, aby bylo možné zobrazit celý název funkce, je zobrazena jenom část, která se do zásobníku vleze, a je ukončena třemi tečkami. V případě najetí kurzorem myši na zásobník, dojde k jeho zvětšení a překrytí části s kódem. Zvětšen je tak, aby byla zobrazena všechna jména v zásobníku celá.



Obrázek 5.4: Snímky výsledného řešení zobrazení nalezeného rozdílu. Na horním snímku je zachyceno zobrazení zdrojového kódu funkce při výběru funkce s nalezeným sémantickým rozdílem ze zásobníku volání. Na dolním snímku je zachyceno zobrazení při výběru volající funkce (v tomto případě porovnávané funkce). Vytvořené řešení má na obrazovkách s nižším rozlišením zmenšený zásobník volání, který se zvětší při najetí kurzoru myši. Nezobrazené části kódu je možné zobrazit kliknutím na „Expand lines“. Jedná se o zobrazení stejného rozdílu, který byl zobrazen na Výpisu 2.1.

Na Obrázku 5.4 jsou snímky vytvořeného řešení zobrazování nalezených rozdílů, kde je možné kromě zásobníku volání vidět také zdrojový kód vybrané funkce.

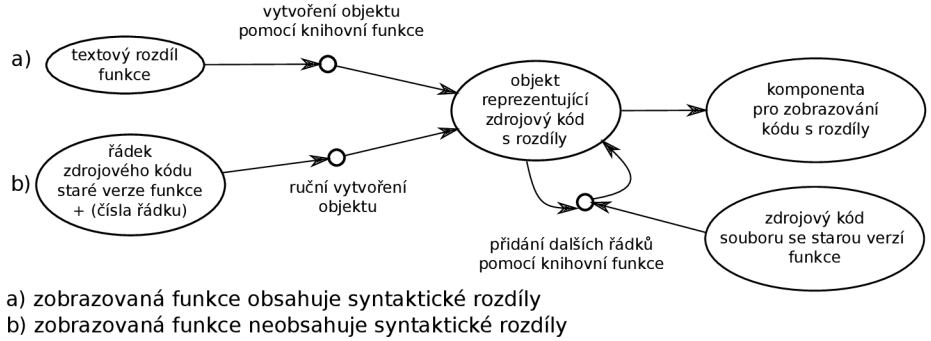
Zobrazení kódu funkce

Pro vybranou funkci ze zásobníku volání jsou uvedeny cesty k souboru, ve kterém se funkce (její stará a nová verze) nachází (požadavek č. 10), také je zobrazen kód funkce. V případě, že funkci není z nějakého důvodu možné zobrazit objeví se upozornění informující o této skutečnosti.

Zobrazení kódu funkce (požadavek č. 6) má na starosti komponenta `DiffViewWrapper`, která k tomu používá vybranou knihovnu react-diff-view. Vstupem (props) komponenty pro zobrazování kódu z tohoto balíčku je objekt, který reprezentuje zdrojový kód s textovými rozdíly (dále jen „objekt“).

- Objekt je možné vytvořit z textového rozdílu (diff) souborů pomocí balíčkem poskytované funkce. Toho je využito v případě, kdy zobrazovaná funkce obsahuje textové (syntaktické) rozdíly. To zda pro funkci existuje textový rozdíl je uchováno v definicích výsledku. Pokud rozdíl existuje, tak je načten ze souboru, který byl vytvořen ve fázi přípravy výsledků (Sekce 5.2).
- V případě, že funkce syntakticky různá není, je nutné objekt používaný komponentou vytvořit ručně. K jeho vytvoření je použit první řádek zdrojového kódu staré verze zobrazované funkce a čísla řádků, na kterých se tento řádek nachází pro starou a novou verzi funkce. Uvedením těchto čísel je zajištěno správné číslování zobrazeného řádku kódu (požadavek č. 9) a také řádků, které budou vloženy (zobrazeny) později. Informace o tom, kde funkce začíná v obou verzích funkce, je uvedena v definicích výsledku. Přístup prohlížeče ke starým zdrojovým souborům, ve kterých se funkce nacházejí, byl zajištěn ve fázi přípravy výsledků (Sekce 5.2).

Jakmile je objekt vytvořen, je možné přidávat další řádky, které se mají zobrazit, pomocí balíčkem poskytované funkce. Tyto řádky jsou získány ze souboru, ve kterém se nacházela stará verze funkce. Obrázek 5.5 se snaží shrnout výše popsané informace.



Obrázek 5.5: Obrázek se snaží zachytit proces zobrazování zdrojového kódu

Zobrazená funkce (vytvořený objekt) je doplněna o její první a poslední řádek. Další prováděné akce jsou závislé na tom, jaká funkce se zobrazuje.

- Pokud je vybrána **funkce s nalezeným sémantickým rozdílem** (poslední funkce ze zásobníku volání), jsou v zobrazeném kódu barevně označeny rozdíly (požadavek č. 4).

- V případě **zobrazení volané funkce** je navíc zobrazen řádek obsahující volání následující funkce ze zásobníku volání. Tento řádek je barevně zvýrazněn (požadavek č. 8). Podle toho, kolik řádků funkce obsahuje, jsou zobrazeny další řádky funkce. Jestliže se jedná o funkci zabírající maximálně 25 řádků tak je zobrazena celá, pokud je rozsáhlejší tak je zobrazeno 10 řádků kolem volané funkce.

K zvýraznění syntaxe v kódu (požadavek č. 7) byla použita knihovna refractor[7]. Pro části kódu funkce, které nejsou zobrazeny, jsou umístěny na tato místa tlačítka, po jejichž stisknutí dojde k zobrazení chybějících řádků.

Instalace balíčků

Proto, aby uživatel nemusel instalovat balíčky používané prohlížečem ručně, je tato instalace součástí manuálního sestavení nástroje DiffKemp, ke kterému se používá nástroj CMake[8]. Pro instalaci je použit npm správce balíčků. Kromě instalace dochází také k sestavení produkční verze aplikace, která je optimalizovaná a minimalizovaná.

[7] refractor – dostupný z https://www.npmjs.com/package/refractor
[8] CMake – dostupný z https://cmake.org/

Kapitola 6

Vyhodnocení řešení

Tato kapitola se zabývá vyhodnocením kvality vytvořeného řešení. Vyhodnocení bylo realizováno pomocí automatizovaného testování vytvořených částí (Sekce 6.1). Kromě toho byla provedena také manuální kontrola (Sekce 6.2), která byla uskutečněna průchodem prohlížeče výsledků pro výsledky analýzy některých verzí jádra RHEL vytvořených nástrojem DiffKemp. Poslední Sekce 6.3 se zabývá srovnáním nového prohlížeče výsledků s původním řešením.

6.1 Automatizované testování

Testy byly vytvořeny pro samotný prohlížeč výsledků (Reactová část nástroje) a také pro část v Pythonu sloužící k vytvoření nového výstupu sémantického porovnání.

Testy prohlížeče výsledků

React aplikace je možné testovat pomocí tzv. end-to-end testů, které testují celou aplikaci většinou v prostředí reálného webového prohlížeče. Dále je možné testovat React komponenty pomocí jednotkových a integračních testů, které testují vykreslený výstup komponenty spíše v jednodušším testovacím prostředí [17].

Testování bylo provedeno *jednotkovými* a *integračními testy*. Byl využit JavaScriptový testovací nástroj Jest[1], který spouští testy, umožňuje vyhodnotit pokrytí kódu a také např. vytvořit *napodobeninu* (mock) komponenty.

Pro samotné testování byla použita knihovna React Testing Library [4], která netestuje React komponenty jako komponenty (jejich vnitřní stav), ale pracuje přímo s vytvořenými uzly (elementy) DOMu[2] a jednotlivé elementy umožňuje získat např. pomocí jejich textu. Tímto chováním se snaží knihovna přiblížit k používání komponenty takovým způsobem, jakým by ji používal konečný uživatel, který by viděl vykreslené HTML elementy [4].

Testovací případy se píší do `it` (nebo `test`) bloku, jednotlivé testovací případy je možné shlukovat do testovacích sad pomocí `describe` bloku. Testovací případ se skládá z:

- přípravy, ve které se nechává vykreslit komponenta pomocí funkce `render`,
- vykonávání akcí, zde dochází k získání elementů, klikání na tlačítko, ...

[1] Testovací nástroj Jest – https://jestjs.io/
[2] Document Object Model – objektová reprezentace HTML dokumentu

- ověření očekávaného chování (zda je element viditelný, má určitou třídu, ...), k tomu slouží funkce `expect` a knihovna jest-dom[3], která slouží pro testování stavu DOMu.

Je možné použít také bloky `beforeEach` a `afterEach`, kód umístěný v těchto blocích se vykoná před každým respektive po každým testu. Blok `beforeEach` byl využit v testovacích sadách pro vykreslení komponenty v případě, že více testovacích případů používalo stejnou komponentu se stejnými vlastnostmi (props).

Pro komponentu `DiffViewWrapper`, která slouží k zobrazení kódu funkce, byly vytvořeny dvě testovací sady:

1. sada testuje případ zobrazení **funkce**, ve které byl **nalezen rozdíl**. Mezi testovací případy této sady patří kontrola:
   - zobrazení řádků, které se ve funkci liší (požadavek č. 4 z Tabulky požadavků 3.1),
   - zobrazení všech řádků funkce po kliknutí na všechna tlačítka, která slouží k zobrazení skrytých (nezměněných) částí kódu.
2. sada testuje zobrazení **funkce s vyznačeným voláním** následující funkce ze zásobníku volání. Zde se kontroluje, že je zobrazen řádek obsahující volání funkce (požadavek č. 8).

V obou sadách se testuje, zda zobrazená čísla řádků a zobrazené řádky kódu odpovídají umístění v souboru, ve kterém se funkce nachází (požadavek č. 9). Při testování bylo využito toho, že komponenta je vykreslena jako tabulka typicky se čtyřmi sloupci (číslo řádku ve starém souboru, text řádku, číslo řádku v novém souboru, text řádku).

Pro komponentu `Callstack` (sloužící k zobrazení zásobníku volání) je testováno zda jsou vyobrazeny všechny funkce, které zásobníky volání (starý a nový) obsahují (požadavek č. 5). To je otestováno pro různé případy, které mohou nastat (např. zásobníky, které mají různý počet volání, některá volání jsou stejná, ale ne všechna).

Vytvořeny byly také integrační testy, které testují komponentu `Difference` (sloužící k zobrazení nalezeného rozdílu) a to např. zda při výběru funkce z komponenty `Callstack` (respektive z vyobrazeného zásobníku volání) dojde k předání správných vlastností (props) komponentě `DiffViewWrapper` sloužící k zobrazení kódu (požadavek č. 6). Toto ověření je zajištěno vytvořením napodobeniny (mock) komponenty – komponenta je nahrazena funkcí, u které se dá kontrolovat s jakými parametry byla volána. Díky tomu lze ověřit, zda by došlo k zobrazení správného kódu. Podobně je testována i celá aplikace, kdy je vytvořena napodobenina komponenty `Difference` a testuje se např. zda jsou ji předány správné parametry při výběru porovnávané a rozdílné funkce pro zobrazení nalezeného rozdílu (požadavek č. 3).

Dále byly vytvořeny jednotkové testy dalších komponent a metod třídy `Result` (obsahuje informace o výsledku analýzy provedené nástrojem DiffKemp) použitých pro výpis funkcí a navigaci ve výsledcích.

Testy Python části

Pro implementaci napsanou v Python části nástroje DiffKemp byly vytvořeny jednotkové testy pro třídu `YamlOutput` (vytvářející výstup ve formátu YAML popisující rozdíly nalezené při analýze nástroje DiffKemp) a třídu `Callstack` (reprezentující zásobník volání od porovnávané funkce k funkci, ve které byl nalezen rozdíl). V případě třídy `Callstack`

[3]Knihovna jest-dom – dostupná z https://github.com/testing-library/jest-dom

jsou testovány jednotlivé metody této třídy. V případě `YamlOutput` je testováno zda ve vytvořeném výsledku (ve formátu YAML) jsou uloženy potřebné informace, které byly reprezentovány v programu.

Testování je provedeno nástrojem pytest[4], který již byl používán pro testování Python části nástroje DiffKemp. Testy se píší do funkcí, které musí začínat slovem `test`. Pro ověření očekávaného stavu se používá příkaz `assert`.

K přípravě prostředí pro testy slouží fixture. Fixtures se píší jako funkce, které mají dekorátor `@pytest.fixture`. Jednotlivé testovací funkce pak mohou přistupovat k dané fixture pomocí zápisu jejího jména mezi své parametry.

V rámci testování třídy `YamlOutput` bylo potřeba vytvořit *fixture* pro objekt reprezentující sestavený výsledek analýzy. K jejímu vytvoření bylo použito *fixture* grafu volání, který byl již používaný jinými testy. Vzhledem k tomu, že nově vytvořené testy byly umístěny do jiného souboru než se nacházeli původní testy s daným *fixture*, bylo potřeba *fixture* grafu volání přesunout do souboru `conftest.py`, který v případě nástroje pytest slouží ke sdílení *fixtures* mezi více moduly.

6.2 Manuální kontrola

Pro ověření kvality vytvořeného řešení byla provedena manuální kontrola prohlížeče výsledků pro nalezené rozdíly nástrojem DiffKemp. Jako zdroj pro nalezení rozdílů byly vybrány tři páry (8.0/8.1, 8.1/8.2 a 8.2/8.3) verzí jádra RHEL. Ten byl vybrán z důvodu toho, že se jedná o primární zaměření nástroje DiffKemp (viz Kapitola 2). Dané verze byly vybrány, protože to jsou nejnovější verze, na kterých se testuje nástroj DiffKemp.

Návrh způsobu ověření

Způsob kontroly je navržen tak, že pro některé nalezené rozdíly dojde ke kontrole shody zobrazeného rozdílu prohlížečem výsledků s původním výstupem (prezentací) nástroje DiffKemp. Samotná **kontrola** by měla být zaměřena na následující části:

- zda zobrazený **zásobník volání** obsahuje všechna jména uvedená v původním výstupu a zda jsou ve správném pořadí (požadavek č. 5 z Tabulky 3.1),
- zda **pro** jednotlivé **volané funkce je vyznačen řádek**, na kterém jsou volány (požadavek č. 8,)
- zda **vyznačený rozdíl** odpovídá rozdílu v původním výstupu (požadavek č. 4),
- zda je **zobrazen zdrojový kód funkcí** (požadavek č. 6) s důrazem na správné číslování řádků (požadavek č. 9) a zobrazení umístění souboru (požadavek č. 10). K tomuto ověření by měly být použity zdrojové kódy porovnaných verzí.

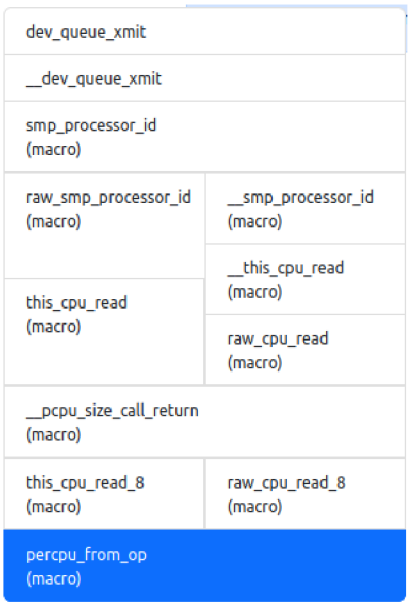
Aby kontrola probíhala i na rozdílech, které jsou v něčem neobvyklé, byla provedena analýza nalezených rozdílů. Ta byla uskutečněna na novém výstupním formátu fáze sémantického porovnání, který díky tomu, že je ve formátu YAML, je dobře analyzovatelný. Byly objeveny následující **speciální případy** (pozn. pojmem funkce se v tomto případě myslí funkce, datový typ nebo makro):

- rozdíly obsahující zásobníky volání, které mají odlišný počet funkcí,

[4]Testovací nástroj pytest – dostupný z https://docs.pytest.org/

- rozdíly obsahující zásobníky volání se stejným počtem volaných funkcí, ale některé funkce nemají stejné jméno,
- funkce, které byly přemístěny do jiného souboru.

Provedená analýza, jak již bylo zmíněno, odhalila případy, kdy zásobníky mají různý počet volaných funkcí. Na Obrázku 6.1 je možné vidět zobrazení jednoho z těchto zásobníků prohlížečem výsledků.



Obrázek 6.1: Snímek reálného zásobníku volání zobrazeného prohlížečem výsledků. Zásobník se vyskytuje v rozdílech mezi verzemi 8.2 a 8.3 jádra RHEL. Stará a nová verze zásobníku obsahuje různé volané funkce (v tomto případě makra) a navíc zásobníky mají odlišný počet volaných funkcí.

Průběh kontroly a nalezené nedostatky

Pro každý pár ze zmíněných tří párů ověřovaných verzí byla uskutečněna kontrola následovně:

- Z každé skupiny *speciálních případů* byla provedena kontrola pro jeden rozdíl. Dále došlo ke kontrole dalších 5 náhodně vybraných rozdílů. Pro tyto rozdíly byla kontrola zaměřena na části, které jsou popsány výše v návrhu.
- Kromě toho bylo navíc prohlédnuto alespoň dalších 20 rozdílů, u kterých došlo pouze ke kontrole toho, že se zobrazuje zdrojový kód a jsou vyznačena volání.

Již samotnou analýzou výsledků bylo zjištěno, že pro jednu datovou strukturu (vyskytující se v nalezených rozdílech) nebude možné zobrazit její zdrojový kód ani rozdíl z důvodu chybějící informace o jejím konci. Původní výstup také neobsahoval syntaktický rozdíl této struktury. Jedná se o chybu způsobenou nedokonalou metodou pro vyhledávání konce datových typů.

Při kontrole rozdílů bylo zjištěno, že se nezobrazuje zdrojový kód ani syntaktický rozdíl v případě, že se jedná o zobrazení makra. Je to z důvodu chybějících informací o makrech týkajících se jejich definic (soubory, kde jsou umístěny, řádky, kde začínají a končí), z tohoto důvodu není možné vytvořit jejich textový rozdíl ani je následně zobrazit. V kontrolovaných zásobnících se jména maker zobrazovala v pořádku. Také v případech, kdy se makro změnilo na funkci nebo obráceně není možné zobrazit rozdíl a to ze stejného důvodu, v tomto případě ani původní výstup rozdíl nezobrazoval.

Mimo zmíněné problémy s makry v kontrolovaných rozdílech:

- nebyla nalezena chyba v zobrazování zásobníků volání,
- pro většinu (nedostatek č. 1 a 2) volaných funkcí byl správně vyznačen řádek, na kterém byly volány,
- vyznačené rozdíly odpovídaly těm v původním výstupu (výjimka – nedostatek č. 3),
- zdrojový kód vypadal v pořádku včetně číslování řádků až na určité případy (nedostatky č. 4 a 5).

Nalezeny byly následující nedostatky:

1. Existují případy, kdy ve funkci není vyznačena volaná funkce. Problém je v tom, že funkce, která by měla být vyznačena se nevyskytuje v dané funkci, ale až v určité funkci volané z této funkci. Příčinou problému je, že v zásobníku (i v původním výstupu) chybí jedna volaná funkce, protože byla typicky inlinována.
2. Existují vyznačené řádky s voláním, na kterých se nevyskytuje název funkce, která je jako následující v zásobníku volání, ale podle údaje v původním výstupu je vyznačen správný řádek.
3. V případě, že syntaktický rozdíl funkce má jako poslední řádek ve změně (tzv. hunk) prázdný řádek, tak nedochází k zobrazení tohoto řádku. Jedná se pravděpodobně o chybu (bug) vybrané knihovny pro zobrazování rozdílů (react-diff-view).
4. U některých funkcí nemusí být zdrojový kód zobrazen od úplného začátku funkce – může chybět např. předcházející řádek obsahující návratový typ funkce. Toto je způsobeno nedokonalou informací o začátcích funkcí, kterou má nástroj DiffKemp.
5. Ve speciálním případě není zobrazen úplně správný zdrojový kód funkce, místo toho je zobrazeno volání makra. Jedná se o případ, kdy je výsledná funkce vytvořena jako inline funkce až při překladu pomocí makra.

6.3 Výhody nového prohlížeče oproti původnímu řešení

Na závěr jsme provedli obecné srovnání původního způsobu prezentace nalezeného rozdílu s novým způsobem. Účelem bylo zjistit, zda nové řešení prezentuje výsledky přehledněji, respektive zda uživateli usnadňuje práci při jejich prohlížení (analýze). Následuje popis prezentace obou způsobů a zmínění výhod nového řešení:

- V původním výstupu byly pro porovnané funkce vytvořeny soubory obsahující pro ně nalezené rozdíly. V daném souboru pak byly za sebou vypsány funkce, ve kterých byl nalezen rozdíl, s daným rozdílem. Nalezený rozdíl se skládá ze zásobníku volání

a syntaktického rozdílu funkce. Nově je možné výsledky zobrazit pomocí vytvořeného prohlížeče ve formě webové aplikace. Prohlížeč zobrazuje seznam porovnaných funkcí, po výběru některé z nich se zobrazí názvy rozdílných funkcí a následně až nalezený rozdíl. Toto výrazně **zrychluje prohlížení výsledků** z DiffKempu, protože nyní uživatel nemusí hledat jim požadovaný rozdíl mezi rozdíly ostatních funkcí.

- Prohlížečem zobrazený zásobník volání obsahuje stejně jako původní názvy jednotlivých volaných funkcí. Oproti původnímu obsahuje také název porovnané funkce, jelikož jednotlivé názvy slouží k zobrazení kódu dané funkce, ve kterém je vyznačen řádek obsahující volání následující funkce. Na rozdíl od původního výstupu je zobrazen jen jeden zásobník, který je rozdělen v místech odlišně volaných funkcí. Díky této vlastnosti může uživatel **snadno poznat, zda došlo ke změně zásobníku** volání mezi porovnávanými verzemi.

- Stejně jako původní výstup, zobrazuje i prohlížeč syntaktický rozdíl ve funkci s nalezeným sémantickým rozdílem. Oproti původní prezentaci umožňuje prohlížeč navíc zobrazit i nezměněné části této funkce. Což umožňuje prohlížet **kontext, ve kterém je rozdíl nalezen**.

- V původním výstupu byl u jednotlivých volání v zásobníku zapsán soubor a řádek, na kterém dochází k volání dané funkce. V případě prohlížeče výsledků je toto řešeno trochu jiným způsobem a to vyznačením řádku s volanou funkcí v kódu volající funkce. Jméno souboru je zobrazeno nad zobrazeným kódem. Původní výstup zdrojový kód funkcí nezobrazoval. Nyní díky němu má uživatel snadnější přístup k širšímu **kontextu v jakém je funkce** s nalezeným rozdílem **používána** porovnávanou funkcí.

- Oproti původní prezentaci výsledků umožňuje nové řešení (pomocí navigace) jednoduše zobrazit další porovnávané funkce, které používají funkci s nalezeným sémantickým rozdílem. Díky tomuto je mnohem **jednoduší zjistit, jak změna** v konkrétní funkci **ovlivní** např. **sémantiku funkcí** tvořících rozhraní knihovny. V případě původní prezentace by bylo nutné pro stejné zjištění projít všechny soubory s nalezenými rozdíly a zkoumat, zda se v nich nevyskytuje jméno rozdílné funkce.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit přehlednější prezentaci výsledků nástroje DiffKemp (sloužícího k nalezení sémantických rozdílů mezi verzemi programů), jelikož jeho původní podoba prezentace byla uživatelsky nepřívětivá – ukládána do textových souborů.

Po seznámení se s nástrojem DiffKemp a jeho způsobem prezentace výsledků bylo zjištěno, že klasické aplikace sloužící k zobrazování rozdílů (jako např. Meld) jsou k prezentování výsledků nástroje nedostačující. Zobrazují celé soubory se všemi syntaktickými změnami, kdežto pro nástroj DiffKemp jsou důležité pouze změny ve funkcích s nalezeným sémantickým rozdílem. Navíc tyto aplikace neumožňují zobrazit relevantní kontext (zásobníky volání) a vztahy mezi analyzovanými funkcemi, které nástroj poskytuje.

Následně byl navržen nový způsob prezentace – prohlížeč výsledků realizovaný ve formě webové aplikace s využitím JavaScriptové knihovny React a frameworku Bootstrap. Došlo k výběru knihovny pro zobrazování rozdílů – vybrán byl balíček react-diff-view, který poskytuje větší možnosti výběru toho, co se bude zobrazovat. Dále bylo navrženo a realizováno uživatelské rozhraní prohlížeče s cílem umožnit uživateli snadnou navigaci v nalezených rozdílech a jejich přehledné zobrazení. Za účelem snadnějšího zpracování výsledků prohlížečem byl navržen a vytvořen nový výstup fáze sémantického porovnání nástroje DiffKemp, který poskytuje výsledky a další užitečné informace v serializované podobě. Díky tomu je možné nalezené rozdíly nyní i poměrně jednoduše analyzovat. Také proběhla refaktorizace funkce nástroje DiffKemp pro vytváření textových rozdílů s kopírovaným kontextem funkcí a doplnění o vytváření rozdílů se sjednoceným kontextem, jelikož ten potřebuje vybraná knihovna.

Vytvořené řešení bylo vyhodnoceno pomocí automatizovaného testování a také manuální kontrolou na shodu zobrazení rozdílu prohlížečem výsledků s původní prezentací nástroje. Na závěr bylo provedeno srovnání původního způsobu prezentace s nově vytvořenou, která oproti původní umožňuje rychlejší navigaci ve výsledcích, usnadňuje uživateli rozpoznat, zda nedošlo ke změně v zásobnících volání, umožňuje jednodušeji zjistit dopad změny ve funkci na další porovnávané funkce a navíc oproti původnímu výstupu zobrazuje zdrojové kódy funkcí.

Vytvořený prohlížeč by se dal v budoucnu rozšířit o zobrazování rozdílů a zdrojových kódů maker. Dále by bylo možné ještě více usnadnit navigaci ve výsledcích a to např. pomocí vyhledávání funkcí. Prohlížeč by mohl být také schopen zobrazovat nalezené výsledky pro více verzí současně nebo umožnit uživateli interakci v podobě např. „schvalování“ (určení závažnosti) nalezeného rozdílu.

Literatura

[1] Bootlin. The Elixir Cross Referencer. *GitHub* [online]. Listopad 2022 [cit. 23. 1. 2023]. Dostupné z: https://github.com/bootlin/elixir.

[2] *Comparing and Merging Files* [online]. Free Software Foundation, Inc., Listopad 2008 [cit. 16. 4. 2023]. Dostupné z: https://www.gnu.org/software/diffutils/manual/diffutils.html.

[3] Decker, K. jsdiff. *GitHub* [online]. Červenec 2022 [cit. 23. 1. 2023]. Dostupné z: https://github.com/kpdecker/jsdiff.

[4] Dodds, K. C. a přispěvatelé. React Testing Library. *Testing Library* [online]. © 2018-2023. revidováno 9. 8. 2022 [cit. 6. 4. 2023]. Dostupné z: https://testing-library.com/docs/react-testing-library/intro.

[5] *Draft: JSX Specification* [online]. Meta Platforms, Inc., © 2014 [cit. 7. 2. 2023]. Dostupné z: https://facebook.github.io/jsx/.

[6] Kiefer, M., Klebanov, V. a Ulbrich, M. Relational Program Reasoning Using Compiler IR: Combining Static Verification and Dynamic Analysis. *Journal of Automated Reasoning*. Září 2017, sv. 60, s. 337–363. DOI: 10.1007/s10817-017-9433-5.

[7] Lahiri, S. K., Hawblitzel, C., Kawaguchi, M. a Rebêlo, H. SYMDIFF: A Language-Agnostic Semantic Diff Tool for Imperative Programs. In: Madhusudan, P. a Seshia, S. A., ed. *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 712–717. ISBN 978-3-642-31424-7.

[8] Learn React. *React Docs (Beta)* [online]. Facebook, © 2023 [cit. 6. 2. 2023]. Dostupné z: https://beta.reactjs.org/learn.

[9] LLVM Project. *LLVM Language Reference Manual* [online]. © 2003–2022, Revidováno 15. 12. 2022 [cit. 3. 1. 2023]. Dostupné z: https://llvm.org/docs/LangRef.html.

[10] Malík, V. DiffKemp. *GitHub* [online]. Prosinec 2022 [cit. 2. 1. 2023]. Dostupné z: https://github.com/viktormalik/diffkemp.

[11] Malík, V. a Vojnar, T. Automatically Checking Semantic Equivalence between Versions of Large-Scale C Projects. In: *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2021, s. 329–339.

[12] MDN přispěvatelé. JavaScript. *MDN Web Docs* [online]. © 1998–2023. revidováno 13. 12. 2022 [cit. 9. 2. 2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/.

[13] *Npm Docs: Documentation for the npm registry, website, and command-line interface* [online]. 2023 [cit. 9. 2. 2023]. Dostupné z: https://docs.npmjs.com/.

[14] Ravi, P. React Diff Viewer. *Npm* [online]. Květen 2020 [cit. 23. 1. 2023]. Dostupné z: https://www.npmjs.com/package/react-diff-viewer.

[15] Rival, X. a Yi, K. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. The MIT Press, 2020. ISBN 978-0262043410.

[16] Ryder, B. Constructing the Call Graph of a Program. *IEEE Transactions on Software Engineering*. 1979, SE-5, č. 3, s. 216–226. DOI: 10.1109/TSE.1979.234183.

[17] Testing Overview. *React* [online]. Meta Platforms, Inc., © 2023 [cit. 6. 4. 2023]. Dostupné z: https://legacy.reactjs.org/docs/testing.html.

[18] Watzke, D. Unixové nástroje – 8 (diff a patch). *AbcLinuxu* [online]. Únor 2010, [cit. 3. 2. 2023]. Dostupné z: https://www.abclinuxu.cz/clanky/navody/unixove-nastroje-8-diff-a-patch.

[19] Willadsen, K. Visual diff and merge tool. *Meld* [online]. © 2021 [cit. 3. 2. 2023]. Dostupné z: https://meldmerge.org/.

[20] Zhang, G. react-diff-view. *GitHub* [online]. Listopad 2022 [cit. 23. 1. 2023]. Dostupné z: https://github.com/otakustay/react-diff-view.

Příloha A

Obsah přiloženého paměťového média

Soubory a složky na přiloženém médiu jsou následující:

- `/diffkemp/` – Zdrojové soubory nástroje DiffKemp.
- `/diffkemp/rhel-kernel-get` – Open-source nástroj[1] pro automatické stažení a přípravu verzí jádra Linuxu.
- `/tex/` – Zdrojové soubory této práce.
- `/README.md` – Readme soubor obsahující tuto přílohu a návod pro spuštění.
- `/analyser.py` – Vytvořený skript, který byl použit pro analýzu nalezených rozdílů.
- `/xpetrl06-Vizualizace-vysledku.pdf` – Tato bakalářská práce v PDF.

V této práci byly vytvořeny (implementovány) následující části nacházející se ve složce `/diffkemp/`:

- `view/` – Složka se zdrojovými kódy vytvořeného prohlížeče výsledků.
- `diffkemp/output.py` – Soubor, který obsahuje implementaci vytváření nového výstupu sémantického porovnání (YAML) – třída `YamlOutput`.
- `diffkemp/diffkemp.py` – V tomto souboru bylo ve funkci `compare` přidáno volání vytvořené třídy `YamlOutput`. Také se zde nachází implementace příkazu `view` (fáze přípravy výsledků) a to ve funkci `view`.
- `diffkemp/semdiff/result.py` – V tomto souboru došlo k refaktorizaci – vytvoření třídy `Callstack`, která nahradila původní reprezentaci zásobníku volání v podobě řetězce.
- `diffkemp/syndiff/function_syntax_diff.py` – V tomto souboru došlo k refaktorizaci existující funkce `syntax_diff` a vytvoření funkce `unified_syntax_diff`, která slouží k vytváření textových rozdílů se sjednoceným kontextem pro funkce.
- `tests/unit_tests/result_test.py` – Soubor s vytvořenými testy pro Python části nástroje DiffKemp, které byly přidány.

[1] rhel-kernel-get – nástroj je dostupný z https://github.com/viktormalik/rhel-kernel-get/

Příloha B

Manuál

Pro použití nástroje DiffKemp je nutné mít nainstalované potřebné závislosti tohoto nástroje a také nástroje `rhel-kernel-get`. Závislosti jsou popsány v README souborech ve složkách jednotlivých nástrojů. Jedná se o následující závislosti:

- `gcc`, `make`, `cpio`, `tar`, `xz`, `bzip2`, `rpm2cpio`, `cscope`, `cmake`, Ninja, `bison`, `flex`, `libelf-dev`, `libssl-dev`
- Clang a LLVM (podporovaná verze 9–15)
- Python 3 s CFFI (`python3-cffi`)
- Python balíčky ze souboru `/diffkemp/rhel-kernel-get/requirements.txt` a `/diffkemp/requirements.txt`
- Node.js (14), npm (8)

Pro zbytek tohoto návodu se předpokládá vykonávání příkazů ze složky `/diffkemp/`. Po instalaci závislosti je nutné nástroj DiffKemp sestavit pomocí následujících příkazů:

```
mkdir build
cd build
cmake .. -GNinja -DCMAKE_BUILD_TYPE=Release
ninja
cd ..
pip install -e .
```

Manuální kontrola a obecné srovnání

Pro zopakování manuální kontroly a obecného srovnání nově vytvořeného řešení s původním je nutné nejprve získat verze jádra Linuxu, pro které pomocí nástroje DiffKemp najdeme rozdíly mezi jednotlivými verzemi a poté je pomocí vytvořeného prohlížeče výsledků budeme moci zobrazit. Pro získání verzí jádra Linuxu slouží nástroj `rhel-kernel-get`. Následující příkazy stáhnou a nakonfigurují jádra Linuxu (RHEL) verze 8.0, 8.1, 8.2 a 8.3

```
rhel-kernel-get/rhel-kernel-get.py -o kernel --kabi 4.18.0-80.el8
rhel-kernel-get/rhel-kernel-get.py -o kernel --kabi 4.18.0-147.el8
rhel-kernel-get/rhel-kernel-get.py -o kernel --kabi 4.18.0-193.el8
rhel-kernel-get/rhel-kernel-get.py -o kernel --kabi 4.18.0-240.el8
```

Následně je nutné pomocí nástroje DiffKemp vytvořit snapshoty pro jednotlivé verze:

```
bin/diffkemp build-kernel kernel/linux-4.18.0-80.el8/ \
snapshots/linux-4.18.0-80.el8 \
kernel/linux-4.18.0-80.el8/kabi_whitelist_x86_64
bin/diffkemp build-kernel kernel/linux-4.18.0-147.el8/ \
snapshots/linux-4.18.0-147.el8 \
kernel/linux-4.18.0-147.el8/kabi_whitelist_x86_64
bin/diffkemp build-kernel kernel/linux-4.18.0-193.el8/ \
snapshots/linux-4.18.0-193.el8 \
kernel/linux-4.18.0-193.el8/kabi_whitelist_x86_64
bin/diffkemp build-kernel kernel/linux-4.18.0-240.el8/ \
snapshots/linux-4.18.0-240.el8 \
kernel/linux-4.18.0-240.el8/kabi_whitelist_x86_64
```

Poté je možné sémanticky porovnat jednotlivé verze mezi sebou (nalézt pro ně rozdíly):

```
bin/diffkemp compare -o diff-linux-4.18.0-80.el8-linux-4.18.0-147.el8/ \
--show-diff snapshots/linux-4.18.0-80.el8/ snapshots/linux-4.18.0-147.el8/
bin/diffkemp compare -o diff-linux-4.18.0-147.el8-linux-4.18.0-193.el8/ \
--show-diff snapshots/linux-4.18.0-147.el8/ snapshots/linux-4.18.0-193.el8/
bin/diffkemp compare -o diff-linux-4.18.0-193.el8-linux-4.18.0-240.el8/ \
--show-diff snapshots/linux-4.18.0-193.el8/ snapshots/linux-4.18.0-240.el8/
```

Nyní jsou v adresáři složky obsahující nalezené rozdíly pro páry verzí jádra Linuxu:

- `diff-linux-4.18.0-80.el8-linux-4.18.0-147.el8/` – rozdíly pro verze 8.0/8.1
- `diff-linux-4.18.0-147.el8-linux-4.18.0-193.el8/` – rozdíly pro verze 8.1/8.2
- `diff-linux-4.18.0-193.el8-linux-4.18.0-240.el8/` – rozdíly pro verze 8.2/8.3

Tyto složky obsahují původní prezentaci (nalezených rozdílů) nástroje DiffKemp (`*.diff`) a nově vytvořený výstup (`diffkemp-out.yaml`).

Následuje popis možného zopakování vyhodnocení pro verze 8.2/8.3, obdobně je možné postup aplikovat i pro ostatní verze. Vytvořený prohlížeč je možné spustit pomocí příkazu:

```
bin/diffkemp view diff-linux-4.18.0-193.el8-linux-4.18.0-240.el8/
```

Po provedení příkazu se zobrazí adresa, kterou je možné vložit do webového prohlížeče a začít prohlížet výsledky. Ty je možné srovnat s původní prezentací. Pro analýzu nalezených rozdílů (pro nalezení „speciálních“ případů) je možné použít přiložený skript:

```
../analyser.py \
diff-linux-4.18.0-193.el8-linux-4.18.0-240.el8/diffkemp-out.yaml
```

Automatizované testování

Vytvořené testy pro Python část je možné spustit příkazem:

```
pytest tests/unit_tests/result_test.py
```

Pro spuštění testů prohlížeče výsledků (Reactová část) je potřebné být ve složce `view`, poté je testy možné spustit příkazem:

```
npm test -- --watchAll
```

Možné je také získat pokrytí kódu a to přidáním přepínače `--coverage`.