VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GENEROVÁNÍ STRUKTUROVANÝCH TESTOVACÍCH DAT

GENERATING STRUCTURED TEST DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ONDŘEJ OLŠÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Olšák Ondřej**
Program: Informační technologie
Název: **Generování strukturovaných testovacích dat**
Generating Structured Test Data
Kategorie: Analýza a testování softwaru

Zadání:

1. Nastudujte běžně používané formáty pro strukturovaná data jako JSON nebo XML. Nastudujte testování softwaru založené na vstupních doménách. Seznamte se s platformou Testos pro podporu automatizovaného testování.
2. Rozšiřte stávající algoritmus implementovaný v platformě Testos pro generování strukturovaných dat o podporu JSON a XML.
3. Implementujte algoritmus pro generování náhodných testovacích stukturovaných dat. Generování testovacích dat by mělo odpovídat zadanému kritériu pokrytí datových elementů.
4. Integrujte detekci a generování testovacích dat do platformy Testos. Ověřte správnost implementované funkcionality pomocí jednotkových a integračních testů.

Literatura:

- Kotyz, J.: Nástroj pro tvorbu obsahu databáze pro účely testování software, 2018. Diplomová práce, FIT VUT v Brně.
- P. Ammann, J. Offutt. *Introduction to Software Testing*, Cambridge University Press, 2008. ISBN 978-0-511-39330-3.
- Domovská stránka platformy Testos. http://testos.org

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cílem této práce je vytvořit nástroj umožňující generovat soubory obsahující strukturovaná data za účelem testování na základě vstupních domén. Tato práce se soustředí na stromově strukturovaná data. Nástroj integruje již dříve implementované nástroje pro tvorbu testovacích dat splňujících uživatelem vybrané kritérium pokrytí vstupních domén. Vytvořený nástroj je schopen generovat sady testovacích souborů ve formátu JSON a XML s testovacími daty, které splňují kritérium pokrytí ECC, BCC nebo PWC.

Abstract

The goal of the bachelor's thesis is to create a tool for generating files with structure data content. The purpose of these files is to be used as test data conforming to testing of program input space. This thesis focuses on tree-structured data. The tool integrates tools implemented previously within Testos framework for generating test data in order to satisfy user-defined coverage criterion. The tool is able to generate a set of files in JSON or XML format containing test data satisfying ECC, BCC, or PWC coverage criterion.

Klíčová slova

generování testovacích dat, testování na základě vstupních domén, strukturovaná data, ECC , BCC, PWC, JSON, XML

Keywords

test data generating, input domain testing, structured data, ECC , BCC, PWC, JSON, XML

Citace

OLŠÁK, Ondřej. *Generování strukturovaných testovacích dat.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Generování strukturovaných testovacích dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Olšák
15. května 2019

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Aleši Smrčkovi, Ph.D. za jeho čas, úsilí a rady, které mi věnoval při tvorbě této bakalářské práce.

Obsah

Kapitola 1

Úvod

Bakalářská práce se věnuje vytvoření nástroje umožňujícího tvorbu sad souborů ve formátu JSON a XML obsahujících strukturovaná data. Vytvořená sada souborů, je primárně určena pro testovací účely, ale její finální využití je plně ponecháno na volbě uživatele. Jedním z příkladů využití vytvořené sady souborů je testování komunikace mezi klientem a severem. Tuto komunikaci je možné nahradit odesíláním jednotlivých souborů ze sady vytvořené tímto nástrojem a to buď na serverovou nebo klientskou stranu a sledovat reakci testované části systému pro jednotlivé zprávy (soubory). Tento postup může pomoci k odhalení takové zprávy, při kterém se testovaná část systému chová nesprávně či nereaguje očekávaným způsobem. Uvedený příklad je pouze jedna z možností využití souborů vytvořených tímto nástrojem pro testovací účely.

Sady souborů jsou vytvářeny tak, aby splňovaly uživatelem vybrané kritérium pokrytí datových elementů. Cílem práce však není navrhnout a implementovat nástroj vytvářející kombinace hodnot parametrů splňujících zvolená kritéria, ale integrace nástrojů, které již existují jako součást platformy Testos [10]. Tyto integrované nástroje jsou použity k tomu, aby se zajistilo vytvoření co možná nejlepší a zároveň nejmenší sady testovacích souborů, která splňuje uživatelem zvolené kriterium pokrytí datových elementů. Součástí výsledné implementace jsou kritéria pokrytí datových elementů Each Choice Coverage, Base Choice Coverage a Pair-Wise Coverage. Nástroj Gestr (Generátor strukturovaných dat) umožňuje tvorbu jak celých sad, tak tvorbu jednotlivých testovacích souborů. Výsledný nástroj bude začleněn do platformy Testos.

Tato bakalářská práce je úzce spojena ze souběžně vznikajícími bakalářskými pracemi Pavla Nováčka a Martina Oháňky a souběžně vznikající diplomovou prací Bc. Dušana Želiara, na které se v následujícím textu odkazuji.

V Kapitole 2 jsou popsány některé existující nástroje pro generovaní souborů se strukturovanými daty, dále technologie použité při tvorbě této práce a stručný popis integrovaných nástrojů. Kapitola 3 vysvětluje na konkrétním příkladu jednotlivá kritéria pokrytí, z nichž některá budou použita ve výsledném nástroji. Kapitola 4 pojednává o návrhu generátoru a jeho začlenění do platformy Testos. V kapitole 5 je popsána implementace výsledného nástroje. Závěrečná kapitola 6 je věnována testování výsledného nástroje a možnosti jeho budoucího rozšíření.

Kapitola 2

Existující nástroje a použité technologie

V této kapitole jsou popsaný některé existující nástroje pro tvorbu souborů se strukturovanými daty. Následuje popis formátu pro ukládání strukturovaných dat (konkrétně JSON a XML) a na závěr jsou zde představeny všechny nástroje, které jsou do výsledného programu integrovány.

2.1 Existující nástroje

Kapitola se zabývá popisem některých existujících nástrojů pro tvorbu strukturovaných dat, které mohou být následně použity k testovacím účelům. Popsané jsou zde nástroje *JSON Generator [18]*, *mocker-data-generator [14]* a *Mockaroo [15]*. Tyto nástroje jsem vybral z toho důvodu, že jsou volně dostupné a dávají uživateli možnost přizpůsobit si strukturu výsledných souborů, která je následně naplněny náhodnými daty, takového typu, který si zvolí sám uživatel.

JSON Generator [18]

Jedná se o webový nástroj umožňující generování souborů ve formátu JSON dle šablony, kterou si může uživatel sám sestavit za použití předdefinovaných klíčových slov (značek) a JavaScriptu. Vygenerované soubory si následně může uživatel stáhnout, může je sdílet popřípadě uložit přímo na server.

Výhody

- Náhodná data u jednotlivých klíčů generovaná na základě zvolených značek.
- Podpora JavaScriptu, který umožňuje tvorbu složitějších konstrukcí.

Nevýhody

- Nelze vytvořit více souborů najednou.
- Podpora pouze formátu JSON.
- Nelze specifikovat kritéria pokrytí datových elementů.

mocker-data-generator [14]

Je knihovna, která spojuje několik různých generátorů náhodných dat. Knihovna umožňuje tvorbu složitějších konstrukcí jako generování dvojice klíč a hodnota, generování dat v cyklu, podpora unikátních klíčů, volba množství prvků v poli nebo objektu, a další.

Výhody

- Open source.
- Integrace více nástrojů generujících soubory do jedné knihovny.
- Možnost tvorby složitých konstrukcí, podpora cyklů a unikátních klíčů.

Nevýhody

- Není zde podpora variant (vytvořená šablona generuje pouze jeden typ souboru s rozdílnými daty).

Mockaroo [15]

Je webový nástroj, který umožňuje generovat soubory v různých formátech jako je JSON, XML, SQL a mnoha dalších. Uživatel si přes grafické rozhraní určí typ jednotlivých dat, jména elementů/klíčů a množství vygenerovaných záznamů. Výsledný soubor si pak může stáhnout nebo zobrazit. Navzdory velkému množství podporovaných souborů je nástroj nejvíce použitelný pro tvorbu obsahu relační databáze. Důvodem je, že záznamy jsou generovány do jednoho souboru a při použití formátu SQL nástroj generuje kompletní příkazy pro vložení dat do databáze.

Výhody

- Jednoduché a přehledné uživatelské rozhraní.
- Podpora velkého množství formátů výstupního souboru.

Nevýhody

- Verze zdarma omezena na 1000 záznamů na soubor.
- Všechny záznamy jsou vygenerovány do jednoho souboru.
- Rychlost generování (lze zlepšit zakoupením licence).

2.2 Použité technologie

XML

XML (Extensible Markup Language) je značkovací jazyk, který byl vyvinut a standardizován sdružením W3C [8] jako podmnožina značkovacího jazyka SGML (Standard Generalized Markup Language). Jazyk XML byl navržen tak, aby na rozdíl od jazyka SGML umožňoval jednoduché strojové zpracování. Je využíván především pro výměnu dat po internetu a zápisu souborů vyžadujících určitou strukturu.

Struktura dokumentu

XML dokument je tvořen jedním a více **elementy**, kde každý element má svůj identifikátor v podobě jména. Volba jména je ponechána na uživateli. Výsledné jméno však musí splňovat určitá pravidla, která budou uvedena dále v textu. Každý element má svoji počáteční a koncovou značku. Pokud však element neobsahuje žádný obsah (jiné elementy, data) může být zapsán pomocí zkráceného zápisu jednou značkou. Každý XML dokument musí obsahovat jeden **kořenový element**, který obsahuje všechny ostatní elementy dokumentu. Aby byl XML dokument validní (well-formed), musí být všechny elementy vzájemně správně zanořeny tzn. pokud je počáteční značka obsažena v jiném elementu, pak musí být koncová značka součástí stejného elementu jako značka počáteční. Toto je jedna z nejdůležitějších podmínek, které musí soubor ve formátu XML splňovat. Ostatní podmínky lze nalézt ve specifikaci [4].

Každá počáteční a zkráceně zapsaná prázdná značka v sobě může obsahovat libovolné množství dat (atributů) v podobě dvojice **jméno** a **hodnota**. Pro jméno atributů a elementů musí platit následující pravidla [11]:

- Jméno atributu se ve značce může vyskytnout pouze jednou.
- Jméno elementu/atributu musí začínat podtržítkem nebo písmenem.
- Jméno elementu/atributu může obsahovat písmena, číslice, pomlčky a podtržítka.
- Jméno elementu/atributu nesmí obsahovat mezery.
- Jméno elementu nesmí začínat písmeny xml, XMLm, xml, ...

Hodnota atributu musí být vždy ohraničena jednoduchými nebo dvojitými uvozovkami např. `<element_name attribute_name="attribute_value">`. Podrobnější popis XML dokumentu a jeho obsahu lze najít na stránkách W3C [4].

Zpracování XML dokumentů

XML dokument lze procházet za pomoci standardizovaného výrazového jazyka **XPath** [12]. Používá cestu (podobně jako v souborovém systému) k navigaci v XML dokumentu. Jeho primárním účelem je adresování částí XML dokumentů. Díky tomu lze jednoduše přistupovat k elementům, jejich atributům, hodnotám atributů a samotnému obsahu elementů.

Další možností jak pracovat se souborem ve formátu XML je za použití DOM (The Document Object Model) [3]. Jedná se o programově aplikační rozhraní (API) pro HTML a XML dokumenty. Definuje způsob jakým přistupovat a manipulovat se soubory ve formátu XML. DOM pracuje s XML dokumentem jako se stromovou strukturou. Vše co je obsaženo v dokumentu (elementy, hodnoty, atributy a dokonce i komentáře) je reprezentováno jako uzel této stromové struktury.

DOM umožňuje:

- Jednoduchou tvorbu XML dokumentů.
- Pohybovat se stromovou strukturou dokumentu.
- Provádět operace nad jednotlivými uzly.

Podrobnější informace o formátu XML a způsobu jeho zpracování lze nalézt na stránkách organizace W3C [4].

JSON

JSON (JavaScript Object Notation) je formát určený pro snadnou výměnu dat. Poprvé byl představen na stránce *json.org* [5] v roce 2001. Jeho popularita stoupala společně s popularitou programovacího jazyka JavaScript. Struktura JSONu dovoluje jednoduché čtení i zápis prováděné člověkem. Lze jej také jednoduše strojově zpracovávat a generovat.

Struktura dokumentu

JSON je tvořen několika konstrukcemi, které se mohou vzájemně zanořovat. Tyto konstrukce jsou:

- **objekt** - neuspořádaná množina dvojic v podobě klíč/hodnota,
- **pole** - kolekce hodnot,
- **hodnota** - řetězec, číslo, pravdivostní hodnota, objekt, pole nebo speciální hodnota 'null'.

Díky tomu, že JSON obsahuje jen tyto jednoduché konstrukce je lehce programově zpracovatelný v každém programovacím jazyce. Soubor ve formátu JSON lze stejně jako XML vyjádřit pomocí stromové struktury. Podobně jako u formátu XML i zde existuje výrazový jazyk (**JSONPath**), který umožňuje procházet strukturou souboru ve formátu JSON. Tento jazyk však narozdíl od XML není standardizován. Na internetu se však objevují různé nestandardizované nástroje umožňující tvorbu JSONPath např. *jsonpath* [16], které je v případě potřeby možno využít. Pro více informací o formátu JSON je možnost nahlédnout do specifikace [6].

2.3 Použité nástroje

Pro docílení tvorby co možná nejlepších sad testovacích souborů se strukturovanými daty byly použity následující nástroje z platformy Testos [10].

Combine

Combine [20] je webový nástroj umožňující tvorbu testovací sady z definovaných vstupních parametrů SUT[1] splňujících pokrytí typu T-Wise. Nabízí také možnost specifikovat omezení platící mezi jednotlivými bloky. Tento nástroj je součásti testovací platformy Testos. Pro podrobnější informace o způsobu implementace je možné nahlédnout do dokumentace nástroje [20].

[1] Zkratka SUT (*z angl. system under test*), označuje část testovaného systému.

Příklad vstupu s omezujícími podmínkami

Mějme definovanou následující jednoduchou funkci pro výpočet podílu:

```
float division(int a, int b, int c){
       return c / (a + b);
}
```

Obrázek 2.1: Jednoduchá funkce pro výpočet podílu.

Chceme vytvořit testovací sadu splňující pokrytí typu Pair-Wise viz. kapitola 3, pro následující vstupní hodnoty parametrů a, b, c:

- $a = [-4, 0, 1]$,
- $b = [-2, 0, 4]$,
- $c = [-1, 0, 1]$.

Jak je na první pohled vidět, některé kombinace parametrů a a b způsobí dělení nulou, které není definováno. Proto vytvoříme omezující podmínky, které zabrání tvorbě těchto nežádoucích kombinací. Tyto podmínky vypadají následovně:

1. $a = -4 \rightarrow b \neq 4$,
2. $a = 0 \rightarrow b \neq 0$.
3. $b = 4 \rightarrow a \neq -4$,
4. $b = 0 \rightarrow a \neq 0$.

Combine-bcc

Combine-bcc [19] je nástroj umožňující tvorbu testovací sady z definovaných vstupních parametrů SUT splňujících pokrytí typu BCC (bázových bloků[2]). Stejně jako dříve zmíněný *Combine* umožňuje zadávat omezení, které musí mezi bloky platit. Tento nástroj je vytvářen v době psaní této bakalářské práce Vladimírem Užíkem, v rámci jeho bakalářské práce a po dokončení bude taktéž součástí platformy Testos.

dbgenx

Dbgenx [17] je nástroj určený ke generování testovacích dat pro relační databáze. Dovoluje uživateli vkládat data přímo do databáze nebo do určeného výstupního souboru. Dokáže generovat data pro všechny základní datové typy (string, int, bool a v omezení míře i float) a také některé speciální typy jako je (date, datetime). Jednou z jeho funkcí je možnost generování dat na základě před-připravených datasetů, které jsou součástí tohoto nástroje. Nástroj dovoluje uživateli specifikovat omezení platící nad generovanými hodnotami a možnost definice datasetů obsahujících požadované hodnoty. Stejně jako předchozí dva nástroje je i dbgenx součástí platformy Testos.

[2]Bázový blok je takový blok, který je vhledem k ostatním nějakým způsobem významný(např. nejčastěji používaný).

Příklad konfiguračního souboru pro nástroj dbgenx

Konfigurační soubor pro nástroj *dbgenx* je uložen ve formátu YAML. Obsahuje informace o tom, jaká data mají být vygenerována, jaká je struktura databáze, jaká omezené platí nad generovanými hodnotami a případně i které datasety se mají použít. Příklad jednoduchého konfiguračního souboru je na obrázku 2.2.

```
include:
  - cities_cs
types:
  city:
    id: seq(0,1000,1)
        city_name: cities_csmp
    visit_date:
        visitDate: dinterval(2018-01-05, 2019-04-06)
shema:
  city_visit:
     id: city.id
     cityName: city.city_name
     dateOfVisit: visit_date.visitDate
constraints:
     city.id > 10
generates:
     - city_visit = 5
```

Obrázek 2.2: Jednoduchá funkce pro výpočet podílu.

Popis konfiguračního souboru

- **includes** - Seznam názvů datasetů.
- **types** - Obsahuje definice datových typů objektů.
- **schema** - Asociativní pole. Klíčem je název tabulky a hodnotou asociativní pole s názvem sloupce a hodnotou z **types**.
- **constraints** - Obsahuje omezení nad hodnotami.
- **generates** - Určení počtu generovaných položek.

2.4 Platforma Testos

Testos (Test Tool Set) je projekt, který si klade za cíl vytvořit sadu automatických testovacích nástrojů. Kombinuje v sobě nástroje pro různé úrovně nebo druhy testování jako je: testování založené na modelech (Model-based), testování založené na požadavcích (Requirement–based), testování uživatelského rozhraní, testování založené na datech (Data-based) a dynamická analýza (Execution-based). Generátor strukturovaných dat vytvářený v této bakalářské práci spadá do kategorie testování založené na datech a po dokončení bude začleněn do této platformy.

Kapitola 3

Testování softwaru založené na vstupních doménách

Informace v této kapitole byly převzaty z knnniy *Interduction to Software Testing* [13]. Vstupní doména je množina obsahující všechny možné vstupy SUT jako jsou parametry funkce nebo vstupy od uživatele. Tato množina se pak dělí do oddílů nebo bloků, které obsahují jednotlivé části domény. Doménu rozdělujeme do oddílů na základě nějakých společných znaků jednotlivých hodnot (charakteristik). Oddíly musí být vzájemně disjunktní a musí obsáhnout všechny hodnoty obsažené v dané doméně.

3.1 Kritéria pokrytí

Na následujícím jednoduchém příkladu budou vysvětleny jednotlivé kritéria pokrytí vstupních domén. Mějme oddíly A, B, C obsahující bloky:

- $A = [x, y]$,
- $B = [1, 2]$,
- $C = [f, g]$.

Each Choice Coverage (ECC)

Pokrytí Each Choice Coverage (pokrytí každého bloku) vyžaduje, aby každý blok ze všech oddílů byl použit alespoň jednou. Nevýhodnou tohoto kritéria je to, že nevyžaduje žádné kombinace hodnot. Na druhou stranu dává testerovi možnost definovat si vlastní kombinace o kterých si myslí, že by mohly být pro daný SUT kritické. Velikost sady je rovná počtu bloků v oddílů s nejvyšším počtem bloků. Výsledná testovací sada pro náš příklad je zobrazena v tabulce 3.1.

	A	**B**	**C**
1	x	1	f
2	y	2	g

Tabulka 3.1: Testovací sada splňující Each Choice Coverage.

All Combination Coverage (ACoC)

U All Combination Coverage (pokrytí všech kombinací) je vyžadováno, aby byly použity všechny kombinace bloků ze všech oddílů. U našeho příkladu to znamená $2 * 2 * 2 = 8$ testů, protože všechny oddíly obsahují dvě hodnoty. Tento typ pokrytí vytváří největší možnou testovací sadu, což může být v některých případech nepraktické. Velikost sady je rovna $\prod_{i=1}^{Q}(B_i)$, kde B_i je počet bloků v oddílech a Q je počet oddílů.

	A	**B**	**C**
1	x	1	f
2	x	1	g
3	x	2	f
4	x	2	g
5	y	1	f
6	y	1	g
7	y	2	f
8	y	2	g

Tabulka 3.2: Testovací sada splňující All Combination Coverage.

Pair-Wise Coverage (PWC)

Pro splnění kritéria Pair-Wise Coverage (pokrytí všech párů bloků) musí být každý blok z každého oddílu zkombinován s každým blokem ostatních oddílů. Testovací sada pro náš příklad je zobrazena v tabulce 3.3.

Výsledné testovací sady je také možné vyjádřit ne jako páry, ale jako n-tice viz. tabulka 3.4. Hodnota '-' znamená, že z daného oddílů může být vybrán libovolný blok.

	A	**B**	**C**
1	x	1	-
2	x	2	-
3	x	-	f
4	x	-	g
5	y	1	-
6	y	2	-
7	y	-	f
8	y	-	g
9	-	1	f
10	-	1	g
11	-	2	f
12	-	2	g

Tabulka 3.3: Testovací požadavky Pair-Wise Coverage.

Pair-Wise Coverage je konkrétní podoba kritéria T-Wise Coverage (TWC) (pokrytí všech T-tic), kde $T = 2$.

U T-Wise Coverage se nevytváří páry hodnot, ale T-tice. Pokud je hodnota T rovna počtu oddílů, pak je TWC ekvivalentní s pokrytím All Combination Coverage.

	A	B	C
1	x	1	f
2	x	2	f
3	x	1	g
4	y	1	g
5	y	2	g
6	y	1	f

Tabulka 3.4: Testovací sada splňující Pair-Wise Coverage zobecněné na trojice hodnot.

Base Choice Coverage (BCC)

U Base Choice Coverage je vybrán z každé charakteristiky bázový blok. Tyto bloky tvoří jeden test obsahující jen bázové bloky. Další testy jsou tvořeny jako kombinace nebázových bloků s každým bázovým blokem. Výhodou tohoto kritéria je volba významných bloků, což umožňuje testerovi se zaměřit na kritické části SUT.

Vybereme z každého oddílu jeden bázový blok například: 'x', '2', 'f'. Tím získáme bázový test (x, 2, f). Výsledná testovací sada pro toto kritérium je uvedena v tabulce 3.5. Velikost výsledné testovací sady je rovna $1 + \sum_{i=1}^{Q}(B_i - 1)$, kde$B_i$ je počet bloků v oddílech a Q je počet oddílů.

	A	B	C
1	x	2	f
2	x	1	f
3	y	2	f
4	x	2	g

Tabulka 3.5: Testovací sada splňující Base Choice Coverage.

Kapitola 4

Návrh nástroje pro generování souborů s testovacími daty

Tato kapitola popisuje celkový návrh nástroje Gestr, jeho začlenění do platformy Testos a následné možnosti jeho využití. Integraci již existujících nástrojů a návaznost na nástroje právě vytvářené. Dále je zde uveden popis vstupního souboru, souboru pro konfiguraci generátoru hodnot *dbgenx* a transformace souborů ve formátu XML do formátu JSON.

4.1 Analýza požadavků

Podporované kritéria pokrytí datových elementů

Jedním z hlavních požadavků na výsledný nástroj, je integrace již existujících nástrojů pro tvorbu testovacích sad, splňující dané kritérium pokrytí datových elementů. Jedním z pokritérií, které bude generátor ve výsledku podporovat je pokrytí všech párů bloků (PWC). Testovací sadu splňující toto kritérium umožňuje tvořit již zmíněný 2.3 nástroj *Combine*.

Další kritérium pokrytí datových elementů, které bude výsledný nástroj podporovat, je pokrytí bázových bloků (BCC). Pro tvorbu takových testovacích sad bude použit právě vytvářený nástroj *Combine-bcc*, zmíněný v kapitole 2.3.

Jako poslední kritérium, které bude nástroj podporovat, je pokrytí každého bloku (ECC). V rámci platformy Testos není nástroj, který by umožnil takové pokrytí generovat. Jeho splnění je však dostatečně jednoduché a tak bude společně s generátorem vytvořen i modul, který bude schopen toto kritérium vytvářet. Tento modul bude přímo součástí výsledné implementace nástroje.

Integrace nástroje dbgenx

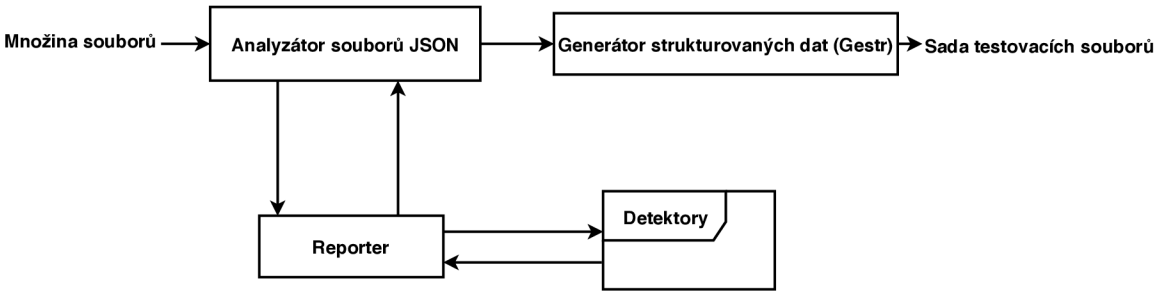
Soubory ve výsledné sadě budou obsahovat náhodná data. V případě souborů ve formátu JSON budou vygenerovaná data použita u klíčů, které jako hodnotu neobsahují pole nebo objekt. V souborech ve formátu XML budou vygenerovaná data použita jako hodnoty u atributů, data elementů, případně jako jména elementů a atributů. Tento požadavek bude splněn za pomoci již představeného 2.3 nástroje *dbgenx*, který byl původně navržen k tvorbě dat pro relační databáze.

Formát výstupních souborů

Mezi podporované formáty výstupních souborů bude patřit JSON a XML. Jelikož bude nástroj převážně pracovat se soubory ve formátu JSON bude třeba provést transformaci souboru ve formátu XML do formátu JSON. Transformace mezi formáty je detailněji popsána v kapitole 4.3. Součásti sady souborů bude i soubor README, ve kterém budou uvedeny informace o vygenerované testovací sadě souborů.

4.2 Návaznost nástroje Gestr na vytvářené projekty

Gestr je jeden ze skupiny právě vytvářených nástrojů, které na sebe navzájem navazují a tím umožňují uživateli snadno a rychle analyzovat obsah velkého množství souborů se strukturovanými daty. Následně pak vytvářet testovací sady souborů, které svojí strukturou odpovídají těmto analyzovaným souborům. Schéma komunikace nástrojů mezi sebou je zobrazena na obrázku 4.1.



Obrázek 4.1: Schéma komunikace mezi nástroji k analýze a generovaní strukturovaných dat.

Analyzátor souborů se strukturovanými daty

Vstupem analyzátoru je množina obsahující N kde $N > 0$ souborů se strukturovanými daty ve formátu JSON nebo XML. Jsou-li na vstupu analyzátoru soubory ve formátu XML je třeba před samotnou analýzou provést jejich transformaci do formátu JSON a to tak, aby byl zachován jejich původní obsah a struktura. Samotná transformace je popsána později v kapitole 4.3. Jakmile množina souborů obsahuje pouze soubory ve formátu JSON můžeme je podrobit analýze za pomoci *Analyzátoru strukturovaných dat* [1]. Analyzátor předává jednotlivé uzly vstupních souborů *Reportéru* za účelem detekce jejich obsahu. *Reportéru* mohou být také předány celé podstromy a to k provedení jejich redukce. Jakmile *Reportér* získá výsledek detekce/redukce, vrátí jej analyzátoru. V případě detekce si analyzátor aktualizuje hodnoty v uzlech, který poslal *Reportéru* k provedení detekce. Pokud byl *Reportéru* předán podstrom k provedení redukce, je analyzátoru vrácen nový podstrom vytvořený na základě toho vstupního. Analyzátor se pak na základě určitých kritérii rozhodne, zda-li bude výsledek redukce akceptovat celý, jeho část, nebo jej nebude brát v potaz. Výstupem analyzátoru je soubor, ve formátu JSON, obsahující abstraktní popis analyzovaných souborů.

Úvod do značek

Pro předávání informací mezi *Reportérem*, detektory a *Generátorem strukturovaných dat* se používají předem dohodnutá klíčová slova neboli značky. Každá značka vyjadřuje informaci o tom, jaká hodnota se v daném uzlu vyskytuje. Značky přidělují uzlům jednotlivé detektory. Seznam podporovaných značek a jejich využití při generování souborů je podrobněji popsán v podkapitole 4.5. Ve zbytku této podkapitoly je popsáno, jak se značkami pracuje *Reportér*, detektory a *Generátor strukturovaných dat*.

Reportér

Jedná se o modul komunikující s *Analyzátorem strukturovaných dat*, který mu jak jíž bylo zmíněno předává jednotlivé uzly nebo celé podstromy k provedení detekce/redukce. Na *Reportér* jsou napojeny jednotlivé detektory umožňující rozpoznávání vybraných typů dat a určování jejich vlastností. *Reportér* na základě již získaných informací zasílá jednotlivé uzly vybraným detektorům, které jsou k němu připojeny. Detektor, kterému se uzel zašle, je vybírán na základě již známých informací o něm (značek). Pokud uzel žádné značky nemá, je volán speciální detektor, který žádnou značku nevyžaduje. Výstupem jsou uzly obohacené o informace (v podobě značek) získané z detektorů případně zredukovaný podstrom.

Detektory

Jak již bylo zmíněno účelem detektorů je určit obsah jednotlivých uzlů a zjištěné informace do něj uložit v podobě značek. Další již zmíněnou operací, kterou zajišťují detektory je redukce podstromů. Každý detektor je nezávislý modul komunikující s *Reportérem*, který jej v případě potřeby aktivuje. Detektory po provedení detekce/redukce vrací výsledek zpět *Reportéru*, který jej buď předá zpět analyzátoru nebo jej předá dalším detektorům. Uživatel může využívat již existující detektory nebo si definovat vlastní detektory, za pomocí předpřipravené šablony.

Generátor strukturovaných dat

Generátor strukturovaných dat na základě vstupních dat vygeneruje sadu testovacích souborů splňující zvolené kriterium pokrytí datových elementů. Vstupem generátoru je soubor vytvořený *Analyzátorem strukturovaných dat*, který obsahuje abstraktní vyjádření souborů, které byly dány na vstup tohoto analyzátoru. Hodnoty, které se budou vyskytovat ve výstupních souborech jsou dány značkami. Tyto značky jsou součástí uzlů reprezentujících hodnotu. Jejich podrobnější popis je uveden v následujících kapitolách.

4.3 Převod formátu XML na JSON

Jelikož je vstupem do nástroje Gestr soubor ve formátu JSON popisující strukturu jiného souboru/ů ve formátu JSON, je třeba všechny soubory ve formátu XML ještě před zahájením jejich analýzy transformovat do formátu JSON, který bude popisovat jejich obsah a strukturu. Při generování se pak nejdříve ve vnitřní paměti vytvoří soubor ve formátu JSON, který se opačnou transformací převede zpět na soubor ve formátu XML, který svojí strukturou odpovídat tomu souboru, který byl na začátku předán k analýze.

Popis převodu mezi formáty

Na obrázku 4.2 je zobrazena ukázka převodu XML elementu s atributy, XML elementu bez atributů a dat v nich obsažených do souboru ve formátu JSON určeného k analýze. Každý element v XML je reprezentován ve výsledném JSONu jako objekt obsahující klíče *name, attributes a children*. Data jsou reprezentována objektem obsahujícím pouze klíč *data*.

- **name** - jméno elementu.
- **attributes** - pole objektů, kde každý objekt reprezentuje jeden atribut elementu a obsahuje klíče:
  - **key** - identifikátor atributu,
  - **value** - hodnota atributu.
- **children** - pole objektů, kde každý objekt reprezentuje jednoho potomka. Objekt v tomto poli musí obsahovat jednu z následujících kombinací klíčů:
  - **name, attributes a children** - pro potomky typu element,
  - **data** - pro potomky typu CDATA (jeho hodnotou je řetězec).

```
<element any_key="data">
  <child_element>
   any_data
  </child_element>
  any_data_2
</element>
```

Příklad XML elementu.

```
{
  "name": "element",
  "attributes": [{
    "key": "any_key",
    "value": "data"
  }],
  "children":[{
      "name": "child_element",
      "attributes": [],
      "children": [{
        "data": "any_data"
      }]
  },{
    "data": "any_data_2"
  }]
}
```

XML element zapsaný ve formátu JSON.

Obrázek 4.2: Nalevo je příklad XML elementu a napravo výsledek transformace do formátu JSON. Pro demonstrační účely nejsou v příkladu zohledněny bílé znaky.

4.4 Popis vstupního souboru

V předchozí podkapitole již bylo zmíněno, že vstupem pro nástroj Gestr je soubor ve formátu JSON, obsahující popis struktury souborů zpracovaných *Analyzátorem souborů obsahujících strukturovaná data*. Tato podkapitola je určena k popisu struktury vstupního souboru nástroje Gestr.

Výsledný návrh je výsledkem spolupráce s Pavlem Nováčkem v rámci tvorby nástroje *Reportér* [9] a Martinem Oháňkou v rámci tvorby *Detektorů* [2]. Oba tyto projekty jsou vyvíjeny ve stejné době jako nástroj Gestr. V této podkapitole jsou popsány jen ty části souboru, které jsou nezbytné pro generovaní výsledné sady souborů. Pro podrobný popis obsahu tohoto souboru je možnost nahlédnout do jedné z uvedených bakalářských prací [9], [2].

Struktura souboru

Každý uzel (objekt, pole, klíč, hodnota), který může soubor ve formátu JSON obsahovat je vyjádřen jako samostatný objekt obsahující dodatečné informace uložené jako hodnoty u předem definovaných klíčů. Tyto hodnoty jsou nezbytné při výběru detektorů, detekci/redukci a výsledném generování sad souborů.

Každý uzel obsahuje z pohledu generování sad souborů dva typy klíčů a to povinné, pro generování nezbytné a nepovinné klíče. Nepovinné klíče jsou většinou takové, které byly vyžadovány v průběhu detekce/redukce, případně sloužily jako informace pro *Reportér*, který na základě nich rozhodoval o volání jednotlivých detektorů. Na činnost generátoru tak nemají žádný vliv.

Klíče, které musí být obsaženy ve všech uzlech

- **type** - obsahuje znak určující, o jaký typ uzlu se jedná:
  - **O** - objekt,
  - **A** - pole,
  - **R** - variantní uzel,
  - **K** - klíč,
  - **V** - hodnota (data).
- **children** - pole objektů, kde každý objekt reprezentuje jeden uzel (potomka).

Uzel typu objekt

V uzlu vyjadřujícím objekt nejsou kromě obou zmíněných povinných klíčů žádné jiné klíče, které by ovlivňovaly výsledek generátoru. Každý vstupní soubor generátoru musí povinně obsahovat jako kořenový objekt uzel tohoto typu.

Pole *children* může obsahovat pouze objekt popisující uzly typu klíč.

```
{
        "type": "O",
        "children": []
}
```

Obrázek 4.3: Zápis uzlu objekt ve vstupním souboru.

Uzel typu Pole

Stejně jako v případě uzlu typu objekt se ani v tomto uzlu nevyskytují žádné jiné povinné klíče ovlivňující generování souborů.

Pole *children* může obsahovat objekty popisující uzly typu: hodnota, objekt, pole a varianta.

```
{
        "type": "A",
        "children": []
}
```

Obrázek 4.4: Zápis uzlu pole ve vstupním souboru.

Variantní uzel

Hodnota R u klíče *type* vyjadřuje, že konkrétní uzel na stejném místě ve stromové struktuře může nabývat více variant. Tyto varianty jsou seskupeny v poli *children*. Při generování je vybrán jeden z potomků tohoto variantního uzlu a tím je vytvořena jedna konkrétní varianta souboru. Způsob výběru potomků je popsán v kapitole 5.7. Uzel neobsahuje navíc žádné jiné než povinné klíče.

Pole *children* může obsahovat objekty popisující uzly typu: hodnota, pole a objekt.

```
{
        "type": "R",
        "children": []
}
```

Obrázek 4.5: Zápis variačního uzlu ve vstupním souboru.

Uzel typu klíč

Objekt reprezentující uzel typu klíč má navíc jeden povinný klíč a to **keyId**, který má jako hodnotu skutečné jméno klíče, který je daným uzlem reprezentován. Pole *children* musí obsahovat alespoň jeden objekt vyjadřující hodnotu u daného klíče, jinak by byl při generování vytvořen klíč bez hodnoty. To by mělo za následek vznik souboru, který by strukturou neodpovídal formátu JSON. Pole *children* může obsahovat:

- $1 - n$ uzlů reprezentujících hodnotu.
- Jeden uzel reprezentující: objekt, variantu nebo pole.

```
{
        "type": "K",
        "keyId": "jmeno_klice",
        "children": []
}
```

Obrázek 4.6: Zápis uzlu klič ve vstupním souboru.

Uzel typu hodnota

Tento uzel reprezentuje konkrétní hodnotu nějakého datového typu (string, integer, float, boolean) nebo hodnotu *null*. Objekt obsahuje navíc povinný klíč *tags*, jehož hodnotou je pole řetězců, kde každý řetězec odpovídá jedné značce. Pole musí obsahovat jednu povinnou značku a to takovou, která určuje datový typ hodnoty, která se v tomto uzlu vyskytuje. Tyto značky jsou: *"string", "int", "float", "bool"* a *"null"*. Seznam některých nepovinných značek je uveden v tabulce 4.1.

Pole *children* by nemělo zůstat prázdné. Jakýkoliv obsah tohoto pole v uzlu reprezentujícím hodnotu, bude při generování ignorován.

```
{
        "type": "V",
        "tags": [],
        "children": []
}
```

Obrázek 4.7: Zápis uzlu typu hodnota ve vstupním souboru

4.5 Generování hodnot

Pro generování hodnot, které budou obsaženy ve výsledných souborech je použit nástroj *dbgenx*. Hodnoty jsou generovány na základě jednotlivých značek. Definice hodnot, které budou generovány na základě těchto značek jsou zapsány v konfiguračním souboru určeném pro nástroj *dbgenx*. Uživateli je umožněno vytvořit si vlastní konfigurační soubor, ve kterém si muže definovat vlastní značky a hodnoty, které se mají na jejich základě generovat. Pokud uživatel nedefinuje svůj vlastní konfigurační soubor, je použit výchozí konfigurační soubor, který je součástí programu a obsahuje definici několika základních značek podporovaných vytvářenými detektory.

V konfiguračním souboru lze specifikovat jména značek, pravidlo, podle kterého bude výsledná hodnota vytvořena a omezení, která musí generovaná data splňovat. Je zde i možnost zvolit si datasety, ze kterých se budou generovaná data vybírat. Tyto datasety jsou součástí nástroje *dbgenx*. Jedná se o soubory ve formátu YAML. Uživatel má možnost si dodefinovat vlastní datasety, za předpokladu, že dodrží strukturu jejich zápisu.

Popis konfiguračního souboru

Na obrázků 4.8 je schéma konfiguračního souboru určeného ke specifikaci generovaných dat pro jednotlivé značky.

- **dataset_name** - obsahuje pole řetězců, kde každý reprezentuje cestu k datasetu ve formátu YAML.
- **types** - obsahuje objekt s klíči, kde každý klíč reprezentuje jméno jedné značky.
- **TAG_NAME** - jméno tohoto klíče určuje jméno značky, která musí být podporována generátorem dat.
- **NAME_OF_VALUE** - jméno tohoto klíče určuje označení hodnoty (označení hodnoty je požadováno nástrojem *dbgennx*).
- **RULE** - pravidlo, podle kterého budou hodnoty vytvářeny. Zápis pravidel je stejný se zapisem pravidel ve vstupních souborech nástroje *dbgenx* [17].
- **constraints** - pole řetězců. Každý označuje omezení, která musí platit nad generovanými hodnotami. Zápis omezení je stejný jako ve vstupních souborech nástroje*dbgenx* [17].

```
{
        "dataset_name": [],
        "types": {
                "TAG_NAME": {
                        "NAME_OF_VALUE": "RULE"
                }
        },
        "constraints": []
}
```

Obrázek 4.8: Schéma konfiguračního souboru pro generování dat.

Omezení platící v konfiguračním souboru

Objekt u klíče **TAG_NAME** musí obsahovat jeden klíč (ve schématu 4.8 vyjádřený jako **NAME_OF_VALUE**), jehož jméno je různé od klíčového slova *ignored* a zároveň může obsahovat nejvýše jeden klíč *ignored*. A to z toho důvodu, že každá značka reprezentuje pouze jednu hodnotu konkrétního typu jako jsou například kladná a záporná čísla. Klíčové slovo *ignored* je vyhrazeno pro hodnoty, které mají být v průběhu zpracování výstupu nástroje *dbgenx* ignorovány. Klíč *ignored* musí být v podobě `"ignored": "int"` povinně obsažen ve všech objektech, ve kterých je generována hodnota typu řetězec. Tato podmínka je způsobena implementací nástroje *dbgenx*.

Pro účely správného fungování nástroje Gestr je třeba zajistit, aby konfigurační soubor obsahoval definici alespoň základních datových typů *(string, bool, float, integer)*. Z tohoto důvodu musí konfigurační soubor v objektu u klíče **types** obsahovat definici značek (klíče) umožňujících generovat alespoň tyto základní datové typy. Ukázka konfiguračního souboru obsahujícího definici pouze povinných značek je zobrazena na obrázku číslo 5.1. Výběr hodnot generovaných u povinných značek je ponechán na uživateli.

```
{
  "dataset_name": [],
  "types": {
        "integer": {
          "integer": "int"
        },
        "float_num": {
          "float_num": "float"
        },
        "string_s": {
          "ignore": "int",
          "string_s": "string"
        },
        "bool_b": {
          "bool_b": "bool"
        }
  },
  "constraints": []
}
```

Obrázek 4.9: Příklad konfiguračního souboru s povinnými klíči v objektu *types*.

Jména značek definovaných ve výchozím konfiguračním souboru jsou vybrána tak, aby odpovídala těm, které je možné detekovat pomoci *detektorů*, které jsou vytvářeny v průběhu tvorby této bakalářské práce. V případě, že v budoucnu budou vytvořeny nové detektory, je možné nové značky dodefinovat do výchozího konfiguračního souboru. Značky podporované ve výchozím konfiguračním souboru jsou zapsány v tabulce 4.1.

Jedna z možností jak blíže specifikovat hodnoty ve výsledných souborech nezávisle na detektorech je definovat si vlastní značku a tu poté ručně připsat k jednotlivým hodnotám, kterých se má týkat. Hodnoty, které jí pak mají být přiřazovány budou specifikovány v konfiguračním souboru nástroje *dbgenx*. V případě konkrétních hodnot může být vytvořen dataset, ze kterého se mají hodnoty pro novou značku získávat.

Název značky	**Popis**
int_pos	kladné celé číslo
int_neg	záporné celé číslo
int_zero	0
float_pos	kladné desetinné číslo
float_neg	záporné desetinné číslo
string_int	celé číslo jako řetězec
string_float	desetinné číslo jako řetězec
bool_true	pravdivostní hodnota True
bool_false	pravdivostní hodnota False

Tabulka 4.1: Seznam výchozích nepovinných podporovaných značek.

4.6 Generování výsledných souborů

Generování souborů ve formátu XML

Kromě již popsané transformace 4.3 souborů ve formátu XML do formátu JSON je třeba zajistit, aby výsledné vygenerované XML soubory odpovídaly (co se struktury týče) souborům předaným k analýze.

Nastavení analyzátoru

Aby bylo možné vytvořit sadu takovýchto souborů, je třeba zabránit ztrátě dat během analýzy. Přesněji zabránit redukcím, které by mohly tuto ztrátu způsobit. Tohoto je možné dosáhnout nastavením *Analyzátoru strukturovaných dat* tak, aby *Reportéru* nebyly zasílány požadavky na redukci, ale pouze požadavky na detekci. Toto nastavení je nad rámec této bakalářské práce.

Značky u uzlů vyjadřujících hodnotu

Dalším problémem, který může mít nežádoucí vliv na výsledné soubory ve formátu XML jsou značky u uzlů vyjadřujících hodnotu. Analyzátor není schopen určit, které uzly vyjadřují jméno XML elementu, jméno atributu, hodnotu atributu a data mezi značkami elementu. Tento problém je možné vyřešit za pomoci speciálních značek nesoucích konkretní hodnotu, která byla uvedena v původním souboru určeném k analýze. Seznam těchto značek je uveden v tabulce 4.2.

Název značky	Popis
i(value)	hodnota typu *integer*
f(value)	hodnota typu *float*
s(value)	hodnota typu *string*
b(true | false)	hodnota typu *bool*
n(null)	konstanta *null*

Tabulka 4.2: Tabulka značek obsahujících konkrétní hodnotu.

Pokud tedy uživatel zadá požadavek na generování souborů ve formátu XML, potom bude v uzlech reprezentujících jméno elementu nebo jméno klíče upřednostňována hodnota uložena v této speciální značce. Pokud však není tato značka přítomna, pak je vygenerována náhodná hodnota, která bude jméno elementu či atributu reprezentovat. Tato hodnota však musí splňovat požadavky na jméno elementu/atributu viz. 2.2.

Generování souborů ve formátu JSON

Při generování souboru ve formátu JSON není třeba pro jména klíčů vytvářet speciální omezení v podobě značek. Jména klíčů jsou totiž v abstraktním souboru obsažena přímo v uzlu, kterým jsou reprezentovány a to v podobě klíče *keyId* viz. 4.6. V případě souborů ve formátu JSON není třeba měnit nějakým způsobem nastavení analyzátoru, jelikož odstranění uzlů nezpůsobí ztrátu dat, která by zabránila generování výsledných souborů.

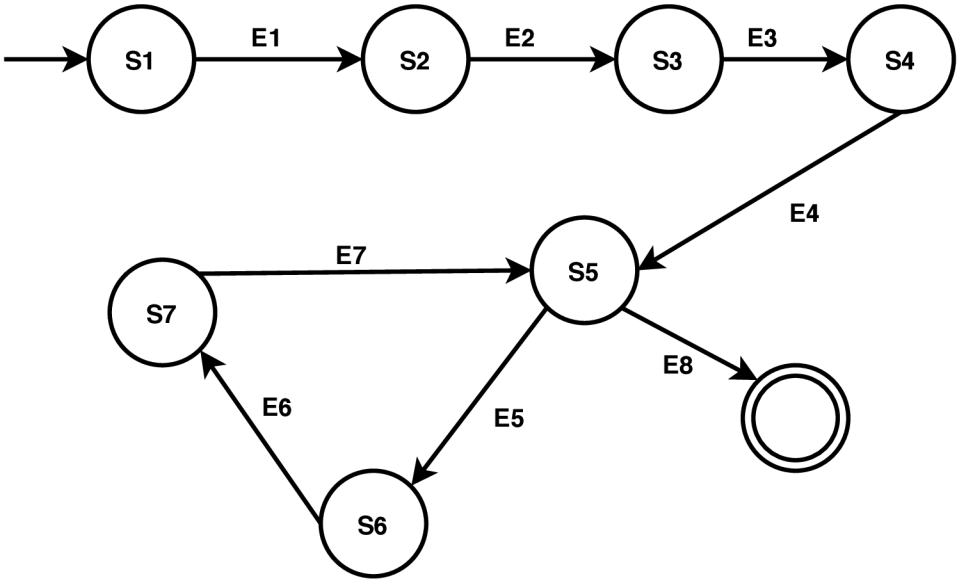
Co se týče značek obsahujících konkrétní hodnotu, je hodnota v nich obsažena přidána mezi ostatní hodnoty vytvořené nástrojem *dbgenx*.

Informace o výsledné testovací sadě

Jedním z požadavků je zaznamenat informace o vytvořené testovací sadě souborů tak, aby kdokoliv, kdo testovací sadu obdrží, dokázal určit, kdy byla vytvořena, jaké kritérium pokrytí datových elementů splňuje, kým byla vytvořena a kolik testovacích souborů celkem obsahuje. Všechny tyto informace jsou uloženy v souboru s názvem *README.md*, který je součástí každé vygenerované testovací sady a to i za předpokladu, že sada bude obsahovat jen jeden soubor.

4.7 Návrh fungování nástroje

Na obrázku 4.10 je zobrazen konečný automat popisující návrh fungování generátoru strukturovaných testovacích dat. V tabulce 4.3 jsou popsány jednotlivé stavy a v tabulce 4.4 jednotlivé přechody tohoto konečného automatu.



Obrázek 4.10: Konečný automat popisující fungování nástroje.

Popis stavů

S1	Načtení souboru obsahujícího abstraktní vyjádření souborů zpracovaných *Analyzátorem strukturovaných dat*. Kontrola, zda-li obsah souboru odpovídá specifikaci formátu JSON.
S2	Vytvoření šablony podle dat uložených ve vstupním souboru. Kontrola výskytu povinných klíčů a jejich hodnot a správného zanoření jednotlivých typů uzlů.
S3	Generování dat pro uzly reprezentující hodnotu za použití nástroje *dbgenx* a jejich následné rozmístění do uzlů.
S4	Tvorba zvoleného pokrytí nad variačními uzly a jejich potomky za využití jednoho z nástrojů *Combine*, *Combine-bcc* nebo modulu pro tvorbu pokrytí každého bloku (ECC), který bude součástí nástroje.
S5	Vytvoření konkrétní cesty šablonou na základě pokrytí vytvořeného ve stavu S4.
S6	Vytvoření stejného typu pokrytí jako ve stavu S4 nad hodnotami jednotlivých uzlů reprezentujících hodnotu.
S7	Vygenerování souborů reprezentovaných cestou vytvořenou ve stavu S5 a kombinacemi hodnot ze stavu S6.

Tabulka 4.3: Popis stavů konečného automatu na obrázku 4.10.

Popis přechodů

E1	Volání metod pro tvorbu šablony z dat uložených ve vstupním souboru.
E2	Výběr všech uzlů reprezentujících hodnotu a volání metody pro generování hodnot na základě jejich značek.
E3	Volání metody pro vytvoření zvoleného kritéria pokrytí nad variačními uzly a jejich potomky.
E4	Volání metody na výběr konkrétní cesty šablonou.
E5	Volání metody pro vytvoření zvoleného kritéria pokrytí nad hodnotami v uzlech.
E6	Volání metody pro generování výsledných souborů.
E7	Opětovné volání metody na výběr konkrétní cesty šablonou.
E8	Ukončení programu, pokud již byly vygenerovány soubory pro všechny vytvořené cesty šablonou.

Tabulka 4.4: Popis přechodů konečného automatu na obrázku 4.10.

Kapitola 5

Implementace nástroje Gestr

Tato kapitola se věnuje popisu implementace nástroje pro generování strukturovaných dat podle návrhu popsaného v kapitole *Návrh nástroje pro generování souborů s testovacími daty* 4.

5.1 Použité technologie

Pro implementaci nástroje byl použit programovací jazyk Python3 ve verzi 3.7. Byl zvolen z toho důvodu, že všechny integrované nástroje jsou v tomto jazyku implementovány, což umožnilo zjednodušit proces jejich integrace do výsledného nástroje.

5.2 Instalace nástroje

Instalace násatroje probíhá za pomocí správce Python balíčků *pip* [7]. V závislosti na nastavení může být nastroj nainstalován pro všechny uživatele, nebo jen pro jednnoho daného uživatele v jeho domovské složce. Během instalace se nainstalují jednotlivé integrované nástroje a knihovny, které vyžadují. Seznam těchto knihoven lze najít v dokumentacích jednotlivých nástrojů [20], [19], [17]. Příkaz pro instalaci nástroje je:

```
python3.7 -m pip install git+https://pajda.fit.vutbr.cz/testos/gestr.git#gestr
```

5.3 Spuštění a ovládání

Po instalaci je možné nástroj ovládat pomocí přepínačů v příkazové řádce. Ukázka spuštění na příkazové řádce:

```
gestr [-h] <-f file> [-d destination_directory] [-o {xml,json}]
       [-c {pwc,ecc,bcc}] [-v {tags,domains}] [-cx file]
```

Popis přepínačů

- **-f** - soubor *.json obsahujícímu abstraktní popis zpracovaných souborů/u. Tento argument je jako jediný povinný.
- **-d** - cesta ke složce, do které bude uložena výsledná testovací sada
- **-o** - volba formátu výstupních souborů (xml, json) výchozím formátem je JSON

- **-c** - volba kritéria pokrytí (pwc, ecc, bcc). Pokud není zadáno je vytvořen jen jeden soubor.
- **-v** - volba způsobu vytvoření hodnot (tags, domains). Pokud není nastaven argument -c pak je hodnota tohoto argumentu ignorována
- **-cx** - cesta ke konfiguračnímu souboru pro *dbgenx*. Pokud není zvolena je použit výchozí soubor.

5.4 Zpracování vstupního souboru

Ihned po spuštění programu, je načten vstupní soubor obsahující abstraktní popis analyzovaných souborů. Je ověřeno, zda-li je soubor skutečně ve formátu JSON a jeho obsah je validní dle specifikace [6] tohoto formátu. Pokud je vše v pořádku, je načtený obsah předán k vytvoření šablony, se kterou se bude po celou dobu běhu programu pracovat.

Šablona je reprezentována jako n-ární strom, ve kterém každý uzel reprezentuje jeden objekt ze vstupního souboru. Při tvorbě této šablony je kontrolován obsah načteného souboru, přesněji povinné značky u uzlů reprezentujících hodnotu a především pak typ potomků v poli u klíče *children*. Mohlo by se totiž stát, že pole *children* bude obsahovat uzel, který nemůže být potomkem daného uzlu.Tato skutečnost by mohla následně vést k vytvoření nevalidní šablony, která by mohla v lepším případě způsobit pád programu, nebo v horším případě generování souborů/u, které by neodpovídaly formátu JSON/XML. Tabulka 5.1 obsahuje informace o tom, jakého typu může mít daný uzel potomky, aby bylo zajištěno vytvoření sady obsahující pouze validní soubory ve formátu JSON, případně XML. Vytvoření šablony má na starosti třída *TemplateCreator*, která v průběhu tvorby šablony kontroluje i typy potomků jednotlivých vytvářených uzlů a také to, že zpracovávaný objekt obsahuje všechny povinné klíče, a jedná-li se o uzel reprezentující hodnotu tak i povinné značky.

Název typu uzlu	**Název typu potomků**
objekt (O)	klíč
pole (A)	hodnota, varianta, objekt, pole
varianta (R)	hodnota, pole, objekt
klíč (K)	objekt, pole, hodnota, varianta
hodnota (V)	-

Tabulka 5.1: Tabulka typů uzlů a povolených typů jejich potomků.

Každý uzel je reprezentován svojí vlastní třídou. Uzly jsou rozděleny do dvou kategorií komplexní a jednoduché. Do komplexních patří uzly reprezentující: *pole, objekt, klíč a variantu*. Všechny tyto uzly jsou potomky třídy *Node*, ve které jsou implementovány všechny operace, které je nad těmito uzly možno provádět. Jmenovitě vkládání potomků a ověřování, že daný uzel je potomkem, ať už přímým, nebo nepřímým uzlu jiného.

Do jednoduchých pak patří ty uzly, které reprezentují konkrétní hodnotu: *bool, float/integer, string a null*. Všechny třídy reprezentující tyto uzly jsou potomkem třídy *ValueNode*, které je potomkem třídy *Node*. Implementuje metody pro zpracování značek a jejich uložení pro další práci s nimi.

Každá třída reprezentující konkrétní typ hodnoty má v sobě definované dvě metody. První *specific_tags* pro práci se značkami, které k získání hodnoty nevyžadují generovaní za pomoci *dbgenx* a tak mohou být hodnoty jimi reprezentované vloženy přímo do pole

hodnot k danému uzlu. A druhou *create_dbgenx_message_by_domains*, která je použita při požadavku na pokrytí domén jednotlivých datových typů. Význam druhé metody je podrobněji popsán v podkapitole 5.5.

Po vytvoření šablony, je metodou *remove_unnecessary_variant_nodes* provedena kontrola, zda šablona neobsahuje uzel typu varianta, který má pouze jednoho potomka. Pokud je takový variační uzel nalezen je z šablony odstraněn. Důvodem je, že by při tvorbě kritéria pokrytí nad variačními uzly a jejich potomky, mohl způsobit chybu v jenom z nástrojů tvořících toto pokrytí. Konkrétně nástroj *Combine* vyžaduje, aby každý parametr měl alespoň dva bloky. Navíc variační uzel pouze s jedním potomkem nemá žádnou funkci a to z toho důvodu, že u něj může nastat pouze výběr jednoho konkrétního potomka.

5.5 Generování hodnot

Po vytvoření šablony jsou vygenerovány hodnoty, definované značkami nebo požadavkem na pokrytí domén datových typů. Dříve již bylo zmíněno, že data jsou generována za pomocí nástroje *dbgenx*. Odesílání požadavku na generování dat má na starosti třída *ValueGenerator*, která kromě odesílání požadavku na generování nástroji *dbgenx* také obsahuje metody na přípravu požadavku a následné zpracování výstupu.

Generování podle značek

Ještě před samotným generováním je metodou *check_config_file* zkontrolována správná definice konfiguračního souboru. Poté jsou z něj metodou *get_generates_names_from_file* vybrána jména značek, které budou při generování hodnot podporovány. Poté jsou metodou *get_tags_from_value_nodes* procházeny všechny uzly typu hodnota a pomocí metody *create_message_by_tags* třídy *ValueNode* je zjištěn celkový počet jednotlivých značek definovaných v konfiguračním souboru. Pro vynechání nedefinovaných značek a pozdějšímu určení jaké hodnoty danému uzlu přiřadit je každá podporovaná značka vložena do pole *required_values*, které je součástí každého uzlu reprezentujícího hodnotu.

Po zjištění počtů jednotlivých značek je poslán generátoru hodnot požadavek na generování přesného typu a počtu hodnot. Po dokončení generování jsou hodnoty rozmístěny do uzlů a to podle jmen značek vložených v poli *required_values*.

Generování pokrytí domén datových typů

Při tomto způsobu generování jsou všechny značky (až na povinný určující datový typ) v hodnotových uzlech ignorovány. Při průchodu uzly je volána již zmíněná metoda *create_dbgenx_message_by_domains*, která hodnoty, pro které nemá smysl volat generátor přímo vloží do pole s výslednými hodnotami daného uzlu. Pro hodnoty, u kterých je vyžadováno použití generátoru je postup stejný jako v případě generování podle značek s tím rozdílem, že značky jsou již předefinované v této metodě. Následující seznam obsahuje hodnoty přiřazené k jednotlivým typům uzlů reprezentujících hodnotu, tak aby byly pokryty domény daného datového typu.

- **String** - null, prázdný řetězec, náhodný řetězec, celé číslo
- **Integer/Float** - null, náhodný řetězec, celé číslo, desetinné číslo
- **Boolean** - null, 0, 1, náhodný řetězec, celé číslo, True, False

- **Speciální uzel *null*** - null, řetězec "null", 0, 0.0, pravdivostní hodnota, náhodný řetězec

Generování hodnot jednoho souboru

Zvláštní případ tvoří generování pouze jednoho souboru, kdy jsou hodnoty generovány na základě datového typu uzlů. To znamená, že jediná značka, která bude použita je značka povinná, určující datový typ hodnoty v uzlu.

Výjimku tvoří případ, kdy generujeme soubor ve formátu XML. Pokud je v poli značek obsažena značka, nesoucí konkrétní hodnotu viz. 4.2 a uzel reprezentující hodnotu vyjadřuje ve výsledném XML souboru jméno elementu nebo atributu, pak je hodnota v této značce použita na místo náhodně vygenerované hodnoty.

5.6 Tvorba pokrytí všech bloků

Vytvoření testovací sady splňující pokrytí všech bloků je vytvářeno ve třídě *EachChoiceCoverage*. Generátor testovacích sad pokrývající všechny bloky je součástí výsledného nástroje a tak byl i návrh a implementace přizpůsobena jeho potřebám.

Tvorba pokrytí ECC nad variačními uzly a jejich potomky

Cílem je, aby se každý potomek každého variačního uzlu vyskytl alespoň v jednom z výsledných souborů testovací sady.

Toho je dosaženo tak, že na vstupu metody *get_combination_of_variant_nodes* tvořící toto pokrytí je pole obsahující všechny variační uzly v šabloně a kořenový uzel šablony. Následně je vytvořeno pole *combination* reprezentující konkrétní případ z testovací sady o stejné délce jako je pole obsahující variační uzly a je naplněno hodnotami $\perp$ (**none** v programu). Toto pole spolu s kořenem šablony a polem variačních uzlů je předáno metodě *go_over_tree*, která rekurzivně prochází šablonou a pokud narazí na variační uzel, rekurzivně se zavolá s tím, že parametr, který byl předtím kořenovým uzlem šablony bude potomek na indexu *actual_child*, který je uložen v každém variačním uzlu (jeho výchozí hodnota je 0) a označuje index potomka v poli *child_list*, který bude vybrán jako následující. Zároveň je do pole *combination* na index odpovídající pozici variačního uzlu v poli *ordered_variant_list* vložena hodnota *actual_child* tohoto variačního uzlu. Pokud uzel není variační je rekurzivně tato metoda volána na všechny jeho potomky. Výsledkem volání metody *go_over_tree* je jeden konkrétní případ z testovací sady splňující pokrytí všech bloků.

Po každém volání metody *go_over_tree* je třeba inkrementovat v některých variačních uzlech hodnotu *actual_child*, a to z toho důvodu, aby mohla vzniknout další část výsledné testovací sady. To provádí metoda *move_variant_node_pointer*. Ta se rekurzivně zanoří až k listovým uzlům a při vynořování inkrementuje u variačních uzlů hodnotu *actual_child* o jedničku a zároveň tento uzel vloží do pole *shifted_nodes*. Inkrementace hodnoty *actual_child* proběhne pouze když jsou splněny následující podmínky:

1. Hodnota *actual_child* je menší jak počet prvků v poli *child_list*
2. Aktuálně zpracovávaný variační uzel není rodičem žádného z uzlů v poli *shifted_nodes*

Volání metod *go_over_tree* a *move_variant_node_pointer* se opakuje dokud návratová hodnota metody *move_variant_node_pointer* není prázdné pole *shifted_nodes*. Jelikož

procházíme šablonou systematicky, není třeba vytvářet omezující podmínky jako v případě nástrojů *Combine* a *Combine-bcc*.

Tvorba pokrytí ECC nad hodnotami

Tvorba pokrytí nad hodnotami je oproti variačním uzlům jednodušší. Nejprve jsou ze všech uzlů reprezentujících hodnotu získány všechny hodnoty (v podobě pole), které mohou nabývat. Poté se hledají uzly, které mají společný klíč. Pokud jsou takové uzly nalezeny, je volána metoda *make_constraints*, která vytvoří nová pole hodnot pro každý z těchto uzlů. Volání teto metody nahrazuje vytváření omezujících podmínek. Nová pole jsou vytvořena tak, že když jeden uzel nabývá hodnoty z původního pole hodnot, tak ostatní uzly nabývají hodnoty $\perp$ (`-` v programu). Výsledná pole pak nahradí ty původní. Výsledek může vypadat třeba takto:

```
# Hodnoty v~uzlech se spolecnym klicem
value_1 = [1,2,3]
value_2 = [4,5,6]

# Puvodni ziskane hodnoty na tvorbu pokryti
values = [[1,2,3],[4,5,6]]

# Nove hodnoty v~uzlech se spolecnym klicem
value_1 = [1,2,3,-,-,-]
value_2 = [-,-,-,4,5,6]

# Nove hodnoty na tvorbu pokryti
values = [[1,2,3,-,-,-],[-,-,-,4,5,6]]
```

Obrázek 5.1: Ukázka práce s hodnotami v uzlech se stejným potomkem typu klíč

Vytváření výsledných kombinací pak probíhá v metodě *create_combinations*. Nejprve se zjistí délka N nejdelšího pole hodnot, která určuje počet cyklů na jeho průchod. V tomto cyklu je další cyklus k procházení jednotlivých polí s hodnotami. Z každého pole je pak vybrán prvek na pozici:

`i % délka aktuálního pole hodnot, kde i je iterátor od 0 do N-1`

Operace modulo zajistí, opakovaný průchod polem s menším počtem hodnot, než má nejdelší pole. To znamená, že nejdelší pole se projde pouze jednou. Tím se zajistí vytvoření nejmenší možné testovací sady.

5.7 Splnění zvoleného kritéria pokrytí datových elementů

Tvorba kritéria pokrytí datových elementů je zajištěna třídami: *BCCCombineInterface* určené ke komunikaci s nástrojem Combine-bcc, *TWCCombineInterface* pro komunikaci s nástrojem Combine a *EachChoiceCoverage*, která přímo implementuje metodu pokrytí všech bloků. Každá z těchto tříd (kromě třídy *EachChoiceCoverage*) obsahuje metody potřebné k vytvoření zprávy pro daný nástroj a převedení výsledku do jednoho formátu tak, aby bylo možně je použít při generování výsledných souborů.

Tvorba vybraného pokrytí nad variačními uzly a jejich potomky

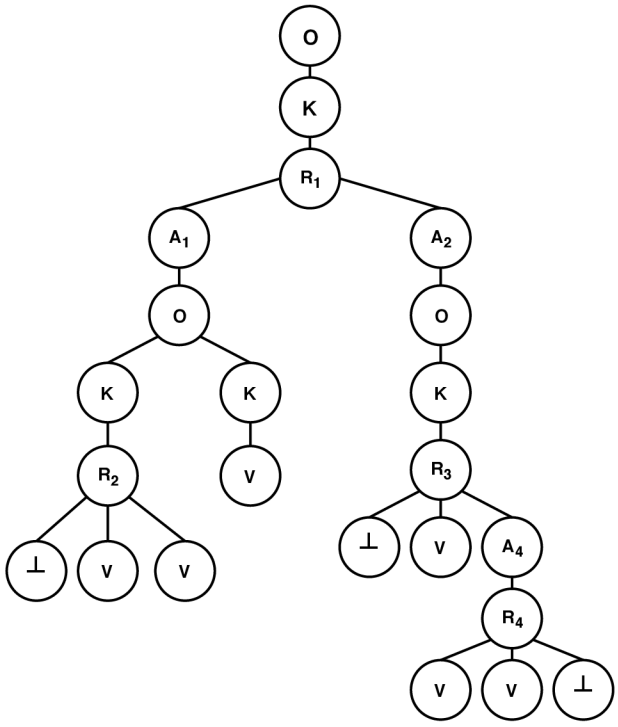
Jediná metoda, která by se v tomto případě z výše uvedených tříd měla používat je metoda *get_combination_of_variant_nodes*. Ta zajistí vytvoření zvoleného pokrytí nad potomky všech variačních uzlů včetně vyloučení nevyhovujících kombinací v podobě *constraints* (podmínek). Výsledné hodnoty jsou pak přiděleny jednotlivým variačním uzlům.

Při tvorbě pokrytí bázových bloku je zde volen jako bázový blok uzel, který je potomkem aktuálně zpracovávaného variačního uzlu a zároveň má nejvyšší počet potomků typu varianta. Pokud má více potomku aktuálního variačního uzlu stejné množství potomků typu varianta, tak je zvolen ten, který má nejvíce potomků různých typů. Pokud mají i stejný celkový počet potomků je z nich vybrán ten, který byl porovnáván jako první.

Vyloučení nevyhovujících kombinací potomků variačních uzlů

Při vytváření bloků, které jsou reprezentované přímými potomky variačního uzlu je ke každému parametru, reprezentujícímu variační uzel, který má alespoň jednoho přímého nebo nepřímého předka typu varianta přidán blok s hodnotou $\perp$, která vyjadřuje, že daný variační uzel nemá být v šabloně vyjadřující jeden konkrétní JSON/XML zahrnut. Tato hodnota je pak využívána při tvorbě podmínek.

Pokud vede cesta z variačního uzlu umístěného v podstromu, vymezeného jedním z potomků rodičovského variačního uzlu do kořene tohoto podstromu, která neobsahuje žádné jiné variační uzly, pak nesmí tento uzel nabývat hodnoty $\perp$. Pokud není variační uzel obsažen v podstromu vymezeném vybraným potomkem variačního uzlu, ale je součástí podstromu s kořenem ve variačním uzlu, který je přímím rodičem aktuálně vybraného potomka, pak jeho hodnota musí být $\perp$.



Obrázek 5.2: Jednoduché schéma šablony.

Ukázka na příkladu

Mějme jednoduchou šablonu 5.2 nějakého vstupního souboru. Jelikož uzel R_1 nemá žádný variační uzel jako rodiče, nemusí obsahovat blok $\perp$.

Určíme kořenem podstromu uzel A_1. Existuje zde cesta z R_2 do A_1, která neobsahuje žádný jiný variační uzel. To znamená, že pokud je vybrán blok s uzlem A_1 nesmí R_2 nabývat hodnoty $\perp$.Ostatní variační uzly nejsou součástí podstromu s kořenem A_1. Jsou však součástí podstromu s kořenem R_1, to znamená, že pokud je vybrán blok s uzlem A_1 pak musí uzly R_3 a R_4 nabývat hodnoty $\perp$. To samé provedeme i s potomkem A_2, kdy uzel R_3 nesmí nabývat hodnoty $\perp$ a uzel R_2 tuto hodnotu nabývat musí. Stejný postup zopakujeme i pro uzly R_3 a R_4.Výsledné podmínky pak jsou:

1. $R_1.A_1 \rightarrow (\neg R2.\ \perp \wedge R_3.\ \perp \wedge R_4.\ \perp)$
2. $R_1.A_2 \rightarrow (R2.\ \perp \wedge \neg R_3.\ \perp)$
3. $R_3.V \rightarrow R4.\ \perp$
4. $R_3.A_4 \rightarrow \neg 4.\ \perp$

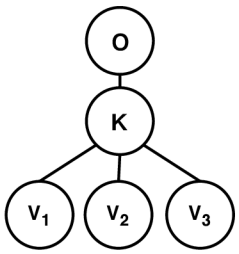
Tvorba vybraného pokrytí nad hodnotami

Při tvorbě pokrytí nad hodnotami uloženými v uzlech reprezentujících konkrétní hodnotu je volána metoda *get_combination_of_value_nodes* implementovaná ve všech zmíněných třídách na tvorbu pokrytí bloků. Výsledkem volání této metody jsou kombinace jednotlivých hodnot splňující uživatelem zvolené kritérium pokrytí bloků.

Při tvorbě pokrytí typu Pair-Wise a Base-Choice coverage jsou hodnoty (bloky) převedeny na řetězce a je k nim přidána informace o jejich původním datovém typu (např. číslo 1 je převedeno na řetězec "int=1"). Tento způsob jsem zvolil proto, že hodnoty u uzlu nemusí být vždy stejného datového typu. Jedním z případů je zmíněné 5.5 generovaní hodnot splňujících pokrytí domén datových typů. Po dokončení tvorby testovací sady a zpracování výsledku jsou tyto hodnoty přetypovány zpět na jejich původní datový typ.

Vyloučení nevyhovujících kombinací hodnot

I při tvorbě kombinací u hodnot je třeba odstranit neplatné kombinace. Neplatné kombinace vznikají, když uzel reprezentující klíč má za potomka více uzlů reprezentujících hodnotu viz. obrázek číslo 5.3. K hodnotám každého uzlu se stejně jako v případě variačních uzlů přidá hodnota $\perp$.



Obrázek 5.3: Jednoduché schéma šablony

Mějme tabulku 5.2 obsahující hodnoty uložené v jednotlivých uzlech na obrázku 5.3. Pokaždé, když vybereme hodnotu z jednoho uzlu která není rovna $\perp$, tak musí všechny ostatní uzly nabývat hodnoty $\perp$.

	uzel V_1	**uzel** V_2	**uzel** V_3
1	string	12	True
2	4,1	string1	$\perp$
3	$\perp$	False	-
4	-	$\perp$	-

Tabulka 5.2: Tabulka hodnot v jednotlivých uzlech

Podle tabulky 5.2 by pak výsledné podmínky (constraints) vypadaly následovně:

1. $(V_1.1 \vee V_1.2) \rightarrow (V_2.4 \wedge V_3.3)$
2. $(V_2.1 \vee V_2.2 \vee V_2.3) \rightarrow (V_1.3 \wedge V_3.3)$
3. $V_3.1 \rightarrow (V_2.4 \wedge V_3.3)$

Pokud děláme kombinace hodnot pouze dvou uzlů, musíme ještě zabránit vzniku případu, kdy oba uzly zároveň budou mít hodnotu $\perp$. Proto by byly vytvořeny navíc ještě dvě podmínky $V_x.\perp \rightarrow \neg V_y.\perp$ a $V_y.\perp \rightarrow \neg V_x.\perp$.

5.8 Generování výsledných souborů

Generování výsledné sady souborů zajišťuje třída *FileGenerator*. Jednotlivé soubory se vytváří rekurzivním procházením šablony vytvořené třídou *TempateCreator*. Výsledný soubor je nejprve reprezentován jako datový typ slovník. Při průchodu šablonou se do slovníku přidá část, kterou aktuálně zpracovávaný uzel reprezentuje. Šablona je procházena metodou *loop_over_tree*, která vrací slovník odpovídající výstupnímu souboru ve formátu JSON.

Uzel typu objekt

Pokud je zpracováván uzel typu objekt, je vytvořen prázdný slovník. Klíče jsou do něj přidávány průchodem pole *child_list* obsahujícího potomky (potomci mohou být pouze typu klíč). Nad každým potomkem je rekurzivně volána metoda *loop_over_tree* a její návratová hodnota je přiřazena ve slovníku k aktuálně zpracovávanému klíči. Výsledný objekt je návratovou hodnotou volání metody *loop_over_tree*.

Uzel typu klíč

U uzlu typu klíč je vrácena hodnota rekurzivního volání metody *loop_over_tree*, které je jako parametr předán první potomek z pole *child_list*.

Variantní uzel

U variantního uzlu je chování stejné jako u uzlu typu klíč, s tím rozdílem, že metodě *loop_over_tree* není jako parametr předán první potomek z pole *child_list*, ale potomek vybraný na základě výsledku vytvořeného kritéria pokrytí variačních uzlu a jejich potomků.

Pozice potomků, kteří budou vybráni z pole *child_list* jsou uloženy ve variačních uzlech v poli *variant_list*. Toto pole obsahuje indexy vyjadřující pozici potomků v poli *child_list*. Při vytvářené konkrétních souborů je (u všech variačních uzlů na stejné pozici) vybrán z pole *variant_list* index potomka, který bude v aktuálně generovaném souboru obsažen.

Uzel typu pole

V tomto případě se vytvoří prázdné pole. Postupně se prochází polem s potomky tohoto uzlu. Nad každým z nich je rekurzivně volána metoda *loop_over_tree* a její návratová hodnota je vložena do dříve vytvořeného pole. Návratovou hodnotou je pak výsledné pole.

Uzel typu hodnota

Uzel reprezentující hodnotu již rekurzivně nevolá metodu *loop_over_tree*, ale vrací konkretní hodnotu. Jedním z parametru metody je také index určující pozici požadované hodnoty v poli *value_list* uloženém v uzlu reprezentujícím hodnotu.

V metodě *select_value* se na základě zvoleného výstupního formátu souborů určí jaká hodnota bude vybrána. Pokud je výstupní formát XML a aktuální uzel je potomkem klíče s *keyId* = *name/key* a zároveň je u hodnotového uzlu definována značka s konkrétní hodnotou viz. 4.2, pak je vybrána hodnota, která byla obsažená ve značce s konkrétní hodnotou. V ostatních případech se zjistí, zda-li předaný uzel neobsahuje na aktuálním indexu v poli *value_list* hodnotu $\perp$. Pokud ne, je hodnota na tomto indexu vrácena. Pokud ano, je volána metoda *get_value_of_next_node*, která prochází pole *value_list* u všech potomků rodičovského uzlu dokud nenajde uzel, který nemá v poli *value_list* na aktuálním indexu hodnotu $\perp$.

Generování jednoho souboru

V případě, že uživatel nezadá jako jeden z parametrů programu typ kritéria pokrytí datových elementů, je výsledkem programu jen jeden soubor. Při tvorbě jednoho souboru se v každém variačním uzlu vybírá jeden náhodný potomek. Tento potomek je kořenem podstromu, který bude celý, nebo jen jeho část (v případě že obsahuje další variační uzly), použit pro tvorbu výsledného souboru. Tvorba hodnot výsledného souboru byla popsána v 5.5. I v tomto případě bude vytvořen soubor *README.md* obsahující informace o výstupu generátoru, s tím rozdílem, že položka *Selected coverage* nebude obsahovat zkratku zvoleného kritéria pokrytí datových elementů, ale hodnotu *one file*.

Generování sady souborů

Samotné vytváření souborů pak probíhá v metodě *generate_outpu_file*, která nejprve ověří existenci zvolené cesty k cílové složce, případně ji vytvoří, a pak do ni začne ukládat výsledné soubory včetně zmíněného souboru *README.md* obsahujícího informace o vygenerovaných souborech.

Pokud je jako výstupní formát zvolen JSON, je použita knihovna jazyka Python *json* a její metoda *json.dump*, které převede slovník do formátu JSON a uloží jej do zvoleného souboru.

Pokud je však jako výstupní formát zvolen XML, je třeba provést dříve popsanou transformaci 4.3 z JSONu do XML, kterou zajišťuje třída *XmlCreator* a její metoda *create_xml_from_json*.

Transformace do formátu XML

V kapitole 4.6 byly popsány značky, které by měli být použity při generování souborů ve formátu XML. Pokud je tato značka u daného uzlu definována a rodičovský uzel reprezentující klíč má *keyId* = *name/key*, pak je hodnota obsažena v této značce při generování souborů ve formátu XML použita přednostně. Je to z toho důvodu, aby zůstali zachovány jména elementů a atributů ve výsledném souboru.

Při transformaci se provádí kontrola jmen elementů a atributů tak, aby splňovaly podmínky zmíněné v podkapitole 2.2. Pokud řetězec začíná číslicí/číslicemi jsou z něj odstraněny. Odstraněny jsou také znaky *xml* pokud jimi řetězec začíná. Může se stát, že nástroj *dbgenx* vrátí jako hodnotu prázdný řetězec, nebo při úpravě jména elementu/atributu vznikne prázdný řetězec. V tom případě je jméno elementu/atributu nahrazeno řetězcem *stringX*, kde X je počítadlo takto nahrazených jmen elementů/atributů. Po provedení kontroly je výsledný obsah v podobě řetězce uložen do souboru.

Ošetření chybových stavů

Jakmile vstupní soubor obsahující abstraktní zápis dříve analyzovaných souborů nebo konfigurační soubor není správně definován (chybějící značky, povinné klíče, ...), je program ukončen a na standardní výstup je uživateli dána stručná informace o důvodu jeho ukončení.

V souboru *src/GestrExceptions* jsou obsaženy definice všech výjimek. Konkrétně obsahuje definice výjimek řešící špatný formát vstupních souborů a případnou chybu, která může nastat v nějakém z integrovaných nástrojů. Téměř všechny tyto výjimky jsou zpracovávány v souboru *___main___*. Důvodem je, že do budoucna bude k nástroji vytvořeno webové rozhraní, díky němuž bude možné ovládat nejen generátor souborů se strukturovanými daty, ale všechny nástroje popsané v podkapitole 4.2. Bude tedy stačit nahradit soubor *___main___* jiným, tvořícím rozhraní mezi nástrojem a uživatelským rozhraním a výjimky dle potřeby zpracovávat v něm.

Kapitola 6

Ověření funkčnosti nástroje Gestr

Výsledný nástroj byl testován pomocí jednotkových testů vytvořených pomocí frameworku *unittest*, který je součástí programovacího jazyka Python. Tyto testy pokrývají 85% celkového kódu. Jednotkové testy jsou použity také k ověření správné komunikace s integrovanými nástroji. Testování komunikace probíhá tak, že je poslán požadavek na daný nástroj a výsledná odpověď je pak porovnána s odpovědí očekávanou.

Pro testování různých kombinací vstupních parametrů byl využit nástroj Combine. Testovací sada pro vstupní parametry je zobrazena v tabulce 6.1.

Volba hodnot	**Typ porytí**	**Výstupní formát**
tags	PWC	JSON
tags	ECC	XML
tags	BCC	JSON
domains	PWC	XML
domains	ECC	JSON
domains	BCC	XML

Tabulka 6.1: Testovací sada vstupních parametrů splňujících pokrytí PWC

Testování nástroje jako jednoho celku pak probíhalo pomocí vytvořených abstraktních souborů, které jsou spolu s nástrojem na přiloženém médiu. Bohužel testovací soubory obsahují jen jednoduché příklady pokrývající kritické části výsledného nástroje. Důvodem je, že analyzátor ještě není v době dokončování této práce hotov a tvorba složitějších souborů by byla velmi časově náročná.

6.1 Rozšíření nástroje

Práce na tomto tématu v sobě ukrývá velké množství různých rozšíření, které mohou být s tímto nástrojem provedeny například v oblasti generování hodnot klíčů, podpory dalších formátu souborů, filtrace uzlů a spousty dalších. Zde jsou podrobněji popsané jen některé z možných rozšíření.

Podpora dalších formátů strukturovaných dat

Nástroj aktuálně podporuje generování souborů ve formátu JSON a XML. Možné rozšíření by mohlo umožnit i podporu pro generování souborů ve formátu YAML. Musela by však

před analýzou být provedena transformace YAML souboru do formátu JSON stejně jako u podporovaného XML. Při tomto převodu by však mohl nastat problém ze ztrátou dat obsažených v původním YAML souboru.

Podpora parametrických značek

Aktuálně podporované značky (kromě značek nesoucích konkrétní hodnotu) v sobě nenesou žádná data, která by více specifikovala jaké hodnoty se mají vyskytnout ve výsledném souboru. Parametrické značky by mohly například určovat výskyt unikátních hodnot jako je například primární klíč. Ve výsledku by se pak mohli hodnoty jednoho souboru odkazovat na obsah souboru druhého. Toho by bylo možné využít například při tvorbě souborů ve formátu JSON určených pro NoSQL databáze.

Filtrování uzlů

Každý uzel by obsahoval váhu odpovídající například počtu výskytů v sadě analyzovaných souborů. Uživateli by pak bylo umožněno zvolit si váhu uzlů, které mají být ve výsledném souboru zahrnuty. Byla by tedy možnost testovat buď běžné soubory vyskytující se frekventovaně nebo soubory obsahující ne příliš časté kombinace hodnot.

Podpora průchodů stromovou strukturou souboru

Jedním z dalších možných rozšíření je podpora JSONPath. To by umožnilo lokalizovat v šabloně konkrétní uzel, se kterým by se následně mohli provádět různé operace např. definování konkrétní hodnoty uzlu, odstranění uzlu nebo jeho podstromu a spousta dalších. Uživatel by si pak mohl definovat cestu k více uzlům společně s operacemi, které se nad nimi mají provádět. Tato pravidla by se uplatňovala jen v případě, když by se uzel vyskytoval v aktuálně vytvořené šabloně.

Tím by bylo uživateli umožněno jednoduše přizpůsobovat obsah výsledných souborů na míru testovanému SUT.

Kapitola 7

Závěr

Výsledný nástroj umožňuje generování testovacích sad souborů ve formátu JSON, splňujících zvolené kritérium pokrytí datových elementů. Podporuje pokrytí bázových bloků (BCC), všech párů bloků (PWC) a pokrytí všech bloků (ECC). Integruje v sobě nástroje tvořící tyto testovací sady a to *Combine* pro splnění kritéria PWC a *Combine-bcc* splňujícího pokrytí typu BCC. Modul pro tvorbu pokrytí všech bloků je součástí samotného nástroje.

Dále integruje nástroj *dbgenx*, který umožňuje generovat data na základě popsaného konfiguračního souboru 4.5, která jsou součástí jednotlivých výstupních souborů. Hodnoty generátor generuje na základě značek. Uživatel může využít již předdefinované značky, nebo si může sám specifikovat vlastní sadu značek za pomoci konfiguračního souboru. Při integraci nástroje *dbgenx* byla zachována podpora datasetů, které mohou být pro generování hodnot využity.

Kromě generování testovacích sad souborů splňujících zvolené kritérium pokrytí datových elementů, umožňuje výsledný nástroj generování jednotlivých souborů s náhodnou strukturou odpovídající abstraktnímu popisu originálních souborů. Při dodržení struktury vstupního souboru je možné výsledným nástrojem generovat i soubory/sady testovacích souborů ve formátu XML.

Nástroj by bylo možné v budoucnu rozšířit o generování specifických hodnot, podporu filtrace uzlů na základě například počtu výskytů ve vstupní sadě souborů, podpory parametrických značek, unikátních hodnot typu primární klíč, podporu průchodů šablonou a s tím spojené ovlivňování dat vygenerovaných ve výstupním souboru/souborech.

Literatura

[1] *Analyzátor souborů se strukturovanými daty*. [Online; navštíveno 23.04.2019]. URL https://pajda.fit.vutbr.cz/testos/treaper

[2] *Automatizovaná detekce datových typů ve strukturách*. [Online; navštíveno 23.04.2019]. URL https://pajda.fit.vutbr.cz/testos/s-detector

[3] *Document Object Model (DOM)*. [Online; navštíveno 02.05.2019]. URL https://www.w3.org/DOM/

[4] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. [Online; navštíveno 02.05.2019]. URL https://www.w3.org/TR/xml/#sec-well-formed

[5] *Introducing JSON*. [Online; navštíveno 24.04.2019]. URL http://www.json.org/index.html

[6] *The JSON Data Interchange Syntax*. [Online; navštíveno 24.04.2019]. URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[7] *The Python Package Index*. [Online; navštíveno 10.05.2019]. URL https://pypi.org/

[8] *The World Wide Web Consortium (W3C)*. [Online; navštíveno 28.04.2019]. URL https://www.w3.org/

[9] *Tree Structure Reporter*. [Online; navštíveno 23.04.2019]. URL https://pajda.fit.vutbr.cz/testos/ts-reporter/tree/dev

[10] Webová stránka projektu Testos. [Online; navštíveno 20.4.2019]. URL http://www.testos.org

[11] *XML Elements*. [Online; navštíveno 02.05.2019]. URL https://www.w3schools.com/xml/xml_elements.asp

[12] *XML Path Language (XPath) 3.1* . [Online; navštíveno 02.05.2019]. URL https://www.w3.org/TR/2017/REC-xpath-31-20170321/

[13] Ammann, P.; Offutt, J.: *Interduction to Software Testing* . Cambridge University Press, 2008, ISBN 978-0-521-88038-1.

[14] Biedma, D.: *mock-data-generator*. [Online; navštíveno 24.04.2019].
URL https://github.com/danibram/mocker-data-generator

[15] Brocato, M.: *Mockaroo*. [Online; navštíveno 24.04.2019].
URL https://mockaroo.com/

[16] Gössner, S.: *Introducing JSON*. [Online; navštíveno 04.05.2019].
URL https://goessner.net/info/

[17] Kotyz, J.: *Nástroj pro tvorbu obsahu databáze pro účely testování software*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018, vedoucí práce Ing.Aleš Smrčka, Ph.D.

[18] Omanashvili, V.: *JSON Generator*. [Online; navštíveno 24.04.2019].
URL https://next.json-generator.com/

[19] Užík, V.: *Base choice coverage criteria test case generator*. [Online; navštíveno 24.04.2019].
URL https://pajda.fit.vutbr.cz/testos/combine-bcc

[20] Červinka, R.: *Asistent pro generování testovacích scénářů*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018, vedoucí práce Ing.Aleš Smrčka, Ph.D.

Příloha A

Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje tyto složky a soubory:

- **examples** - složka obsahující ukázkové vstupní soubory
  - **JSONexamples** - složka obsahující ukázkové abstraktními soubory pro generování sad souborů ve formátu JSON
  - **XMLexamples** - složka obsahující ukázkové abstraktními soubory pro generování sad souborů ve formátu XML
- **gestr** - složka obsahující zdrojová soubory
  - **config** - složka obsahující konfigurační soubor pro generování hodnot
    - **dbgenx_config** - konfigurační soubor pro generování hodnot nástrojem *dbgenx*
    - **README.md** - soubor s popisem konfiguračního souboru
  - **node** - složka obsahující třídy definující uzly
    - **complex** - složka obsahující třídy definující nehodnotové uzly
    - **fundamental** - složka obsahující třídy definující uzly reprezentující hodnotu
  - **__main__.py** - vstupní bod programu
  - **BCCCobmineInterface.py** - soubor obsahující implementaci komunikace s nástrojem *Combine-bcc*
  - **EachChoiceCoverage.py** - soubor obsahující implementaci tvorby pokrytí Each Choice Coverage
  - **FileGenerator.py** - soubor obsahující implementaci generátoru výsledných sad souborů
  - **GestrException.py** - soubor obsahující definici výjimek
  - **TemplateCreator.py** - soubor obsahující implementaci tvorby šablony
  - **TWCCombinneInterface.py** - soubor obsahující implementaci komunikace s nástrojem *Combine*

  - **ValueGenerator.py** - soubor obsahující implementaci komunikace s nástrojem *dbgenx*
  - **XMLGennerator.py** - soubor obsahující implementaci transformace formátu JSON na formát XML
- **test** - složka obsahující soubory s jednotkovými testy
  - **testing_json_files** - složka obsahující abstraktními soubory ve formátu JSON pro účely testování testování
- **install.sh** - instalační script
- **MANIFEST.in** - soubor s informacemi o umístění konfiguračního souboru
- **README.md** - soubor se základními informacemi o nástroji a popisem instalace
- **setup.py** - soubor obsahující informace potřebné pro instalaci