VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO TVORBU OBSAHU DATABÁZE PRO ÚČELY TESTOVÁNÍ SOFTWARE

TEST DATA GENERATOR FOR RELATIONAL DATABASES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAN KOTYZ**
AUTHOR

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

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Kotyz Jan, Bc.**
Obor: Inteligentní systémy
Téma: **Nástroj pro tvorbu obsahu databáze pro účely testování software**
Test Data Generator for Relational Databases
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte relační databáze a testování na základě vstupních domén. Nastudujte utilitu db-gen platformy Testos pro generování strukturovaných testovacích dat. Analyzujte požadavky pro testování systémů pracujících s relačními databázemi.
2. Navrhněte metodu generování kombinací náhodných testovacích dat splňující uživatelem zadaná omezení. Navrhněte metodu generování náhodných testovacích dat pro relační databáze.
3. Implementujte dané metody v samostatném nástroji. Implementujte aplikační rozhraní pro komunikaci s dalšími komponentami platformy Testos (např. s kombinačním generátorem nebo správou testovacích případů).
4. Funkcionalitu generování testovacích dat podpořte automatickými testy.

Literatura:

- P. Ammann, J. Offutt. 2008. Introduction to Software Testing, Cambridge University Press. ISBN 978-0-511-39330-3.
- H. M. Adorf, M. Varendorff. 2014. Constraint-Based Automated Generation of Test Data. In LNBIP, Vol. 166. doi: 10.1007/978-3-319-03602-1_13
- M. Emmi, R. Majumdar, K. Sen: Dynamic Test Input Generation for Database Applications. In Proceedings of Intl. symposium on Software testing and analysis, 151-162, 2007.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato diplomová práce se zabývá problematikou generování testovacích dat pro naplnění obsahu relační databáze. Samotným cílem této diplomové práce je navržení a implementace nástroje, který na základě zadaných omezení umožňuje generovat testovací data. Tento nástroj pro řešení zadaných omezení a samotné generování testovacích dat využívá SMT řešitele.

Abstract

This thesis deals with the problematic of test-data generation for relational databases. The aim of the this thesis is to design and implement tool which meets defined constrains and allows us to generate test-data. This tool uses SMT solver for constraint solving and test-data generation.

Klíčová slova

testování softwaru, databáze, generování testovacích dat, SMT řešitel

Keywords

software testing, database, test-data generation, SMT solver

Citace

KOTYZ, Jan. *Nástroj pro tvorbu obsahu databáze pro účely testování software.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Nástroj pro tvorbu obsahu databáze pro účely testování software

Prohlášení

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

. . . .
Jan Kotyz
23. května 2018

Poděkování

Děkuji svému vedoucímu Ing. Aleši Smrčkovi, Ph.D. za jeho rady, připomínky a čas, který mi při vypracovávání této diplomové práce věnoval.

Obsah

1 Úvod **3**

2 Testování softwaru založené na datech **5**
- 2.1 Testování na základě vstupních domén . . . 5
- 2.2 Kombinační testování . . . 6
- 2.3 Generování testovacích dat . . . 7
- 2.4 Chováním řízený vývoj . . . 8

3 Generování dat pro testování systému využívající databáze **9**
- 3.1 Existující generátory databázových dat . . . 9
  - 3.1.1 Utilita db-gen . . . 10
  - 3.1.2 Další generátory dat . . . 11
- 3.2 SMT řešitelé . . . 11
  - 3.2.1 PySMT – Python knihovna pro práci s SMT . . . 12

4 Návrh nástroje pro tvorbu testovacího obsahu databáze **15**
- 4.1 Specifikace požadavků . . . 15
  - 4.1.1 Požadavky na konfigurační soubor . . . 15
  - 4.1.2 Požadavky na generovaná data . . . 16
  - 4.1.3 Datasety . . . 17
  - 4.1.4 Požadavky na rozhraní . . . 17
- 4.2 Návrh nástroje dbgenx . . . 17
  - 4.2.1 Popis formátu vstupního souboru . . . 18
  - 4.2.2 Načtení dat ze vstupního souboru do vnitřní reprezentace . . . 21
  - 4.2.3 Generování dat podle zadaných omezení . . . 21
  - 4.2.4 Generování dat pro řetězce . . . 24
  - 4.2.5 Generování dat z datasetů . . . 24
  - 4.2.6 Generování dat podle zadaného rozložení pravděpodobnosti . . . 24
  - 4.2.7 Generování podle pokrytí vstupních domén . . . 25
  - 4.2.8 Vkládání dat do databáze . . . 26
  - 4.2.9 Návrh aplikačního rozhraní pro komunikaci s dalšími nástroji platformy Testos . . . 27

5 Implementace nástroje dbgenx **30**
- 5.1 Spuštění nástroje dbgenx . . . 30
  - 5.1.1 Další možnosti použití nástroje dbgenx . . . 31
- 5.2 Použité knihovny a nástroje pro implementaci nástroje dbgenx . . . 31
- 5.3 Implementační detaily zpracování konfiguračního vstupního souboru . . . 31

5.4 Implementační detaily generování dat podle zadaných omezení 31
5.5 Implementační detaily převodu dat na SQL a jeho vkládání do Databáze . . 34

6 Ověření kvality nástroje dbgenx **35**
6.1 Jednotkové testování 35
6.2 Integrační testování 36
6.3 Akceptační testování 36
6.4 Porovnání nástroje dbgenx s předchozím nástrojem db-gen 37

7 Závěr **39**

Literatura **40**

Kapitola 1

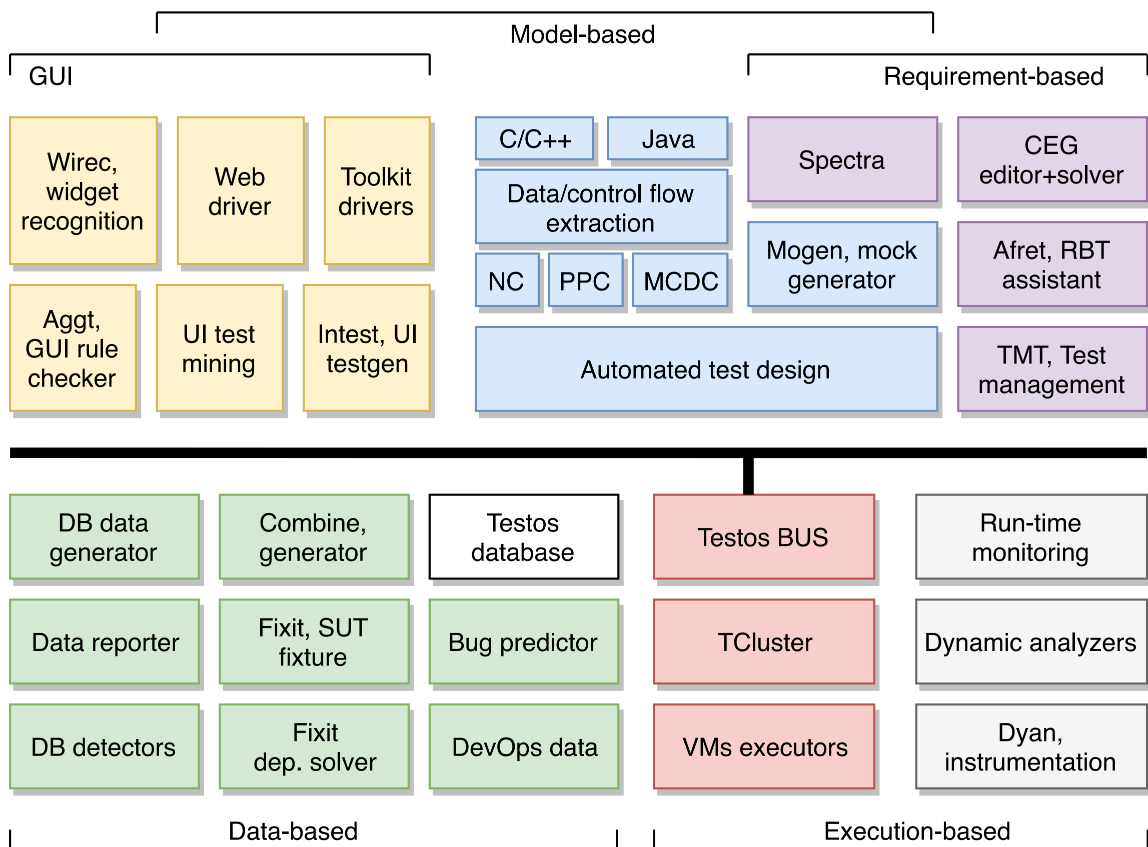
Úvod

Testování je významnou součástí životního cyklu vývoje software a jeho význam neustále roste tím, že se počítače stále více dostávají do běžného života lidí, čímž rostou i požadavky na kvalitu software. S tím také souvisí neustálý vývoj v technikách testování a ve vývoji nových nástrojů pro podporu testování na různých úrovních, které ulehčují život testerům. Jelikož je manuální testování do jisté míry opakující se monotonní práce, je vhodné ji co nejvíce automatizovat. Příkladem může být právě plnění databázového systému daty, na kterých se vyvinutý software bude testovat. Vyplňovat ručně spousty řádků databázových tabulek je zdlouhavé a přitom je možné tuto monotónní činnost automatizovat. Přesně k tomu slouží nástroj představený a vyvinutý v rámci této diplomové práce.

Cílem této diplomové práce je tedy vytvořit pro uživatele, převážně testery, jednoduše ovladatelný nástroj, který umožní naplnění jakékoli relační databáze potřebnými daty vhodnými pro otestování funkcionality testovaného softwaru.

Vytvořený nástroj je součástí platformy Testos (Test Tool Set) [2], což je projekt jehož hlavním cílem je vytvoření sady nástrojů podporující automatizované testování softwaru. Nástroje v platformě Testos (viz obrázek 1.1) kombinují různé úrovně testování a lze je řadit do několika kategorií: testování založené na modelech (Model-based), testování založené na požadavcích (Requirement-based), testování grafického uživatelského rozhraní (GUI), testování založené na datech (Data-based) a dynamická analýza (Execution-based). V aktuálním vývoji nástrojů pro testování založené na datech jsou nástroje pro snadnou tvorbu testovacích dat pro relační databáze: nástroje pro generování náhodných dat [17, 16], nástroje pro zjišťování logických vazeb mezi testovacími daty [10, 12, 15]. Tato diplomová práce primárně navazuje na již existující cizí bakalářskou práci [17].

V kapitolách 2 a 3 jsou popsány teoretické základy přibližující používané technologie, přehled již existujících nástrojů a shrnutí jejich výhod a nevýhod. V kapitole 4 se pak nachází podrobně rozebrané požadavky na vytvářený nástroj pro generování testovacích dat do databáze a návrh, jakým způsobem se budou náhodná testovací data do databází generovat. Dále je popsána samotná implementace v kapitole 5, následována popisem, jak je ověřena kvalita a správnost implementovaného řešení v kapitole 6. V poslední kapitole 7 je shrnuto a zhodnoceno, co bylo v rámci této práce vykonáno, a také jsou představeny návrhy na další možné rozšíření tohoto nástroje.



Obrázek 1.1: Platforma Testos [2].

Kapitola 2

Testování softwaru založené na datech

Testování softwaru obecně slouží k ověření správnosti vyvíjeného produktu a je nedílnou součástí životního cyklu softwaru. V této kapitole je nejprve představeno testování na základě vstupních domén společně se souvisejícím kombinačním testováním. Dále jsou pak popsány možnosti generování testovacích dat a na závěr je krátce představena metodika chováním řízeného vývoje (angl. behaviour driven development, BDD), která byla částečně použita při tvorbě této diplomové práce, kdy je za pomocí testů vyvíjen samotný software.

2.1 Testování na základě vstupních domén

Hlavním cílem kteréhokoliv testování je vždy zvolit nejlepší možné vstupy, nad kterými se následně testy budou spouštět, aby se program vyzkoušel v co nejvíce možných situacích, do kterých by se mohl při svém běhu dostat. Bohužel, ve většině případů není možné v přijatelném čase otestovat program nad všemi možnými vstupy, a tak nastává problém, jak zvolit tu nejlepší vstupní množinu, která by pokryla co nejvíce možností běhu testovaného programu. Jedním z řešení tohoto problému, jak tuto vstupní množinu omezit, a které vstupy by měl případný automatizovaný generátor testovacích dat k testování zvolit, je testování na základě rozdělení vstupních domén do disjunktních bloků.

Jako vstupní doménu můžeme považovat parametry metod a funkcí, globální proměnné, objekty reprezentující současný stav testovaného programu nebo uživatelské vstupy, v závislosti na tom, který softwarový artefakt je právě testován. Vstupní doména je poté rozdělitelná do bloků, kde každý blok obsahuje z pohledu testování stejně užitečné hodnoty a následně jsou pro testování vybírány hodnoty z každého z bloků. [3]

Tyto bloky, tvořící množinu B_q, jsou vždy k sobě vzájemně disjunktní a společně tvoří doménu D. Jedná se tedy o rozklad na její třídy ekvivalence.

$$b_i \cap b_j = \emptyset, i \neq j; b_i, b_j \in B_q$$

$$\bigcup_{b \in B_q} b = D$$

Testování na základě vstupních domén má výhodu v podobě jednoduchosti, kdy k testování není nutné rozumět implementačním detailům, ale vše je založeno jen na základě popisu volby vstupních hodnot. [3]

2.2 Kombinační testování

Při testování jakéhokoliv programového celku většinou nebývá pouze jeden vstup, ale bývá jich povětšinou více (např. více argumentů funkce). Tím nám nevzniká jen problém, které vhodné hodnoty pro jednotlivé vstupy samostatně zvolit, ale k efektivnímu testování je potřeba volit i správné kombinace jednotlivých vstupů. Ve chvíli, kdy máme vstupní domény rozděleny do disjunktních bloků, můžeme uvažovat o možnostech pokrytí jejich kombinací, jak je popsáno v [3]. Níže jsou v této podkapitole představena některá možná kritéria pokrytí kombinací bloků vzniklých rozkladem vstupních domén.

Jedním z kriterií pokrytí, které lze použít, je pokrytí všech kombinací všech bloků (angl. All Combinations Coverage), dále ACoC. Při tomto kritériu pokrytí musí být pokryty, jak už název napovídá, všechny kombinace bloků, což při rozdělení domén do více bloků nemusí být vždy vzhledem k množství kombinací praktické. Kritérium pokrytí **ACoC** si můžeme ukázat na příkladu, kdy tři domény s bloky [A, B], [1, 2, 3], a [x, y] nám vytvoří dvanáct kombinací testovacích případů, viz tabulka 2.1:

	D1	**D2**	**D3**
1	A	1	x
2	A	1	y
3	A	2	x
4	A	2	y
5	A	3	x
6	A	3	y
7	B	1	x
8	B	1	y
9	B	2	x
10	B	2	x
11	B	3	x
12	B	3	x

Tabulka 2.1: Příklad kombinací bloků tří domén splňující kritérium pokrytí ACoC.

Dalším kritériem, které trochu redukuje počet kombinací je kritérium pokrytí každého bloku (angl. Each Choice Coverage), dále ECC. Toto kritérium vyžaduje pouze, aby se mezi testovacími případy vyskytoval každý z bloků alespoň jednou. Jak můžeme vidět na příkladu pro tři domény [A, B], [1, 2, 3], [x, y], ke splnění kritéria ECC nám stačí pouze 3 testovací případy, viz tabulka 2.2:

	D1	**D2**	**D3**
1	A	1	x
2	B	2	y
3	A	3	x

Tabulka 2.2: Příklad kombinací bloků tří domén splňující kritérium pokrytí ECC.

Slabinou tohoto kritéria pokrytí je, že vlastně nevyžaduje žádnou kombinaci hodnot.

Tím se nabízí další kritérium pokrytí a to pokrytí všech párů bloků (angl. Pair-Wise Coverage), dále PWC. Toto kritérium požaduje, aby se mezi testovacími případy nacházely

všechny kombinace dvojice, což pro tři domény [A, B], [1, 2, 3], [x, y] splňuje například následující testovací sada o osmi testovacích případech, viz tabulka 2.3:

	D1	**D2**	**D3**
1	A	1	x
2	A	2	x
3	A	3	x
4	A	-	y
5	B	1	y
6	B	2	y
7	B	3	y
8	B	-	x

Tabulka 2.3: Příklad kombinací bloků tří domén splňující kritérium pokrytí PWC.

Kritérium pokrytí PWC lze zobecnit z pokrytí dvojic na pokrytí všech T-tic bloků (angl. T-Wise Coverage), dále TWC.

2.3 Generování testovacích dat

Automatizace generování testovacích dat je stále jedním z mnoha nedořešených problémů v oblasti výzkumu testování software z důvodu, že se jedná o velice obtížný problém. Nástrojů umožnujících automatické generování testovacích dat není mnoho, ani těch komerčních, a jejich použití bývá značně omezené. [3]

Jedním z možných řešení je generování náhodných vstupních testovacích dat, což je relativně jednoduché na implementaci, ale přináší s sebou i své problémy. Jedním z problémů generování náhodných testovacích dat je, že chyby se v programech nevyskytují rovnoměrně napříč programem, ale vyskytují se více v okolí určitých problematických míst. Chyby v programu se tedy mohou projevovat pouze pro malou množinu hodnot z celkového prostoru všech možných generovaných dat, což může být u náhodného generování dat problém. Dalším z problémů generování náhodných testovacích dat vyvstává při generování strukturovaných dat. Ač je triviální vygenerovat náhodná čísla nebo řetězce posloupností náhodných písmen, ve chvíli kdy mají data svou pevně danou strukturu, tak se náhodné generování dat stává mnohem obtížnějším. [3]

Dalším možným, ač velice komplikovaným řešením je využití technik provádějících nejprve analýzu zdrojového kódu a následné použití analýzou získaných informací ke generování testovacích dat, splňující požadované testovací kriterium. Některé techniky mohou využívat statické analýzy kódu, jiné používají dynamickou analýzu. Problémem těchto technik, založených na analýze zdrojového kódu, mohou být například pokud testovaný program pracuje s přístupy do paměti, kdy nelze předem analyzovat, jaké hodnoty budou mít ukazatele do paměti. Dalším problémem může být tzv. "problém interních proměnných", kdy můžeme mít například požadavek, že hodnota X je větší než hodnota Y na konkrétním řádku zdrojového kódu, ale proměnné X ani Y nejsou mezi vstupními proměnnými. [3]

A na závěr zde uvedených možností řešení je možnost řešení pomocí technik prohledávání stavového prostoru (angl. search-based). Často se používají například genetické algoritmy. Tyto techniky bývají převážně jednoduché na implementaci a bývají také docela flexibilní. Těchto technik bývá užíváno především na úrovni systémového testování. [3]

2.4 Chováním řízený vývoj

Chováním řízený vývoj (angl. behaviour driven development – dále jen BDD) je agilním vývojovým přístupem, jehož obliba v posledních letech roste. Bylo představeno Danem Norhem jako odpověď na nedostatky v testy řízeném vývoji (angl. Test Driven Development – dále jen TDD).[11, 13]

TDD je agilním přístupem, který vyžaduje první vytvořit testy, a až následně vyvíjet. Jedním z problémů TDD ale je, že testy se více zaměřují na ověření stavu testovaného systému, než na požadované chování systému [13]. Oproti tomu BDD je více zaměřeno na chování a vyžaduje podrobnou specifikaci chování vyvíjeného systému, která bude následně automatizovatelná a zároveň pro člověka jednoduše čitelná. K tomu slouží zápis do testovacích scénářů v jazyce Gherkin které mají následující strukturu [11].

> **Given** some initial context (the givens),
> **when** an event occurs,
> **then** ensure some outcomes.

Za klíčovým slovem **Given** je popsáno počáteční nastavení, se kterým je test spuštěn. Následně je za klíčovým slovem **When** popsána nějaká událost a za klíčovým slovem **then** její očekávaný následek.

Pro podporu BDD existují napříč různými programovacími jazyky rozličné frameworky určené pro BDD testování. Níže je uvedeno pár vybraných příkladů těchto frameworků:

- Ruby – Cucumber[1], RSpec[2].
- Java – JBehave[3].
- PHP – Behat[4].
- Javascript – Jasmine[5].
- .NET – SpecFlow[6].

[1] Cucumber – https://cucumber.io
[2] RSpec – http://rspec.info/
[3] JBehave – http://jbehave.org
[4] Behat – http://behat.org
[5] Jasmine – https://jasmine.github.io/
[6] SpecFlow – http://specflow.org/

Kapitola 3

Generování dat pro testování systému využívající databáze

Mnoho současných programů využívá pro svou práci databázi, kterou využívají pro ukládání perzistentních a relativně rychle přístupných dat. Pro správu a práci se samotnou databází pak bývají využívány systémy řízení báze dat (angl. database management system – DBMS), dále jen SŘBD. Správnost a bezchybnost chování samotných programů pak může záviset jak na kvalitě SŘBD, tak na kvalitě samotné programové logice programu. SŘBD bývají povětšinou vyvíjeny a spravovány velkými softwarovými společnostmi a tak lze předpokládat, že pracují správně [8]. Nicméně samotné aplikační logice, která přes SŘBD databázová data využívá, se již tolik pozornosti v ověřování správnosti většinou nedostává [6].

Pro ověření správnosti programů se běžně používá testování, kdy se program opakovaně spouští nad mnoha různými vstupními testovacími daty a kontroluje se, jestli výsledek běhu programu dopadl podle očekávání. Úspěšnost testování a toho zda v programu bude nalezena případná chyba závisí z velké části na tom, jaká vstupní data byla zvolena. A v případě testování systémů pracujících s databází závisí kvalita testování na obsahu databáze.

Jednou z možností, jak testovat systémy pracující s databází je použití živých dat (dat, která v databázi již jsou). Tato možnost ale většinou nebývá tou správnou volbou, protože by testování bylo omezeno pouze na vstupy z aktuální databáze, a zdaleka bychom nepokryli celou vstupní doménu. Také živá data nemusí být pro testování dostupná, ať již z důvodu, že žádná ještě nebyla vytvořena, nebo z důvodu bezpečnosti, kdy by testeři neměli mít přístup k osobním údajům uživatelů. [6]

Druhou možností je pak naplnění databází umělými daty, čímž se zabývá i tato diplomová práce. V této kapitole jsou nejprve představeny již existující generátory dat, využitelné pro naplnění obsahu databází daty. Dále se pak v této kapitole nachází stručné představení SMT řešitelů, a nastínění, jak mohou být SMT řešitelé využití ke generování testovacích dat.

3.1 Existující generátory databázových dat

V současné době již existuje spousta nástrojů, které umí nějakým způsobem automaticky generovat testovací data. Každý z těchto existujících nástrojů má ale nějaké nevýhody nebo nedostatky, které jsou rozebrány v následujících podkapitolách.

V první podkapitole 3.1.1 je představena utilita db-gen, která vznikla jako bakalářská práce [17] a na kterou tato semestrální práce navazuje. V druhé podkapitole 3.1.2 jsou potom krátce představeny některé další existující nástroje pro generování testovacích dat.

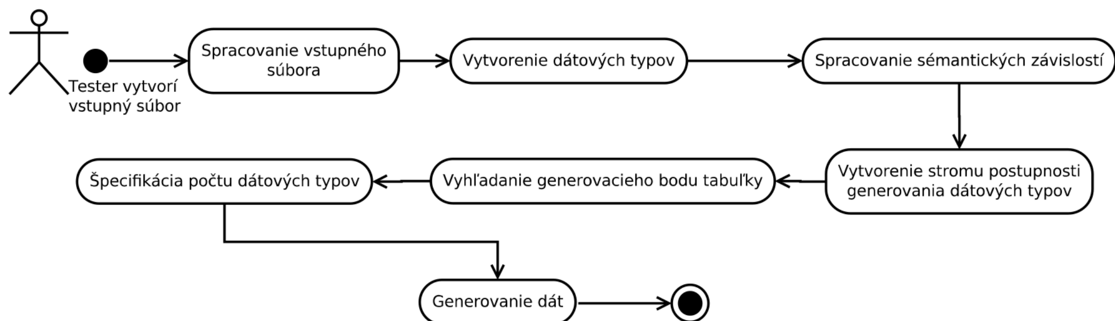
3.1.1 Utilita db-gen

Utilita *db-gen* je nástrojem, implementovaným jako konzolová aplikace, který do databáze generuje náhodná testovací data zohledňující vnitřní omezení databází, pro která jsou tato data generována.

Pro spuštění nástroje *db-gen* je nejprve nutno nastavit parametry v konfiguračním souboru, který se předává na vstupu. V tomto konfiguračním souboru je potřeba specifikovat následující položky.

- Definice nových datových typů, které představují schéma testované databáze.
- Vazby mezi jednotlivými tabulkami (datovými typy) a jejich sloupci.
- Sémantické omezení jednoduchých datových typů.
- Požadované počty, kolik objektů daných typů se má vygenerovat.
- Volitelné připojení již předdefinovaných datových typů a datasetů.

Po nastavení konfiguračního souboru a spuštění nástroje, probíhá běh programu v následujících fázích, jak je popsáno v diagramu na obrázku 3.1.



Obrázek 3.1: Diagram s fázemi vývoje programu [17].

Tento nástroj má ovšem i některé slabé stránky.

1. Hlavní z těchto slabin je použitý náhodný generátor, který vždy náhodně vygeneruje objekt požadovaného datového typu a následně zkontroluje, jestli takto vygenerovaný objekt odpovídá zadaný sémantickým omezením. Pokud ne, pokusí se objekt náhodně vygenerovat znovu. To aby se generátor při nemožné, nebo moc složité specifikaci sémantických omezení nezacyklil je ošetřeno omezením jen na 100 pokusů náhodných generování pro jeden objekt. Po 100 neúspěšných pokusech tedy generování objektu skončí neúspěchem, přestože by bylo možné hodnoty odpovídající sémantickým omezením nalézt, ale díky náhodnosti na ně generátor nenarazil.

2. Dalším omezením tohoto nástroje je, že umí data vkládat pouze do MySQL databází, pro ostatní databázové systémy je potřeba si nejprve nechat vygenerovat INSERT příkazy jazyka SQL do výstupního souboru a následně tyto příkazy zavolat nad požadovanou databází.
3. A dále omezené množství předdefinovaných datových typů a datasetů.

3.1.2 Další generátory dat

- **Generatedata** [1] – Jedná se o opensource grafický webový nástroj, který je pod licencí GPLv3. Data jsou generována za pomoci omezeného množství předem definovaných generátorů. Nástroj umožňuje data generovat do různých výstupních formátů, jako např. CSV, JSON, SQL. Nevýhoda tohoto nástroje pro použití naplnění databáze testovacími daty je v tom, že nijak nezohledňuje schéma databáze, a tak je potřeba data generovat pro každou databázovou tabulku zvlášť a samostatně řešit závislosti mezi tabulkami. Výhodou je, že je nástroj zdarma.
- **Mockaroo** [2] – Další grafický webový nástroj, tentokrát již komerční. Oproti Generatedata umožňuje navíc nastavit poměr kolik hodnot bude prázdných v daném sloupci, a použití některé z nabízených formulí k úpravě generovaných hodnot. Nástroj opět umožňuje exportovat data do různých výstupních formátů. Nevýhodou je omezení na vygenerování pouze 1000 řádků, poté už je nutné zakoupit některý z nabízených placených plánů. Nástroj opět nijak nezohledňuje databázové schéma.
- **SQL Data Generator** [3] – SQL Data Generator je komerčním nástrojem společnosti Red Gate, který je určen pouze pro operační systémy Windows. Výhodou tohoto nástroje je, že dokáže pracovat se schématem databáze a propojovat tak například tabulky pomocí cizích klíčů. Nevýhodou potom je omezení pouze na Windows a SQL Server a komerčnost tohoto nástroje.
- **ApexSQL** [4] – Jedná se o další z komerčních nástrojů, opět fungující pouze na operačních systémech Windows.

Výčet nástrojů není zdaleka konečný, podobných nástrojů jistě existuje mnohem více, než bylo představeno ve výčtu výše. V této podkapitole byly krátce představeny některé nástroje sloužící pro generování dat obecně, i pro generování dat přímo do databází. O žádném z těchto nástrojů ovšem nejde říci, že by byl univerzálně použitelný pro každého. Ať už z důvodů komerčnosti, toho že nástroje nejsou multiplatformní nebo některých jejich dalších omezení.

3.2 SMT řešitelé

Problém splnitelnosti v dané teorii (angl. satisfiability modulo theories – dále jen SMT), je rozšířením problému splnitelnosti (angl. boolean satisfiability problem – dále jen SAT) o rovnost, lineární aritmetiku, bitové vektory, teorie polí a o další použitelné teorie predikátové logiky prvního řádu [7, 9]. SMT řešitel je potom nástroj, který rozhoduje o splnitelnosti

[1] Generatedata – http://www.generatedata.com/
[2] Mockaroo – https://www.mockaroo.com/
[3] SQL Data Generator – https://www.red-gate.com/products/sql-development/sql-data-generator
[4] dbForge – https://www.apexsql.com/sql_tools_generate.aspx

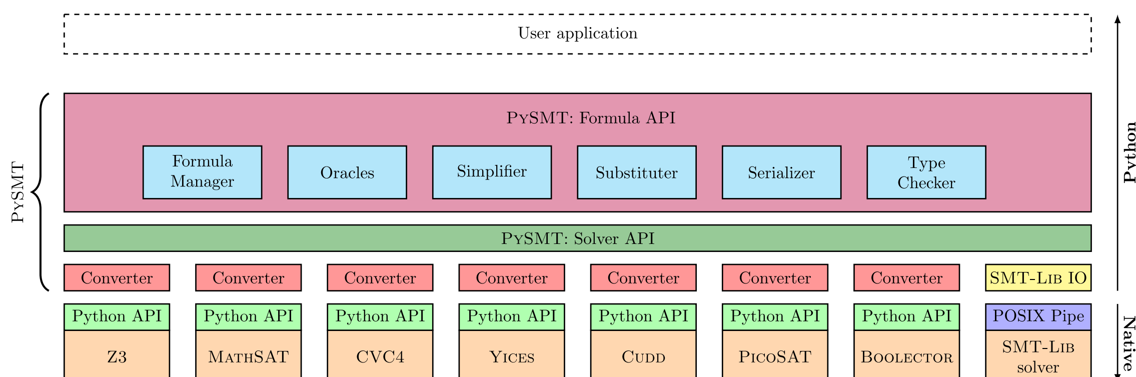
formulí ze zmíněných teorií, což umožňuje SMT řešitelům použití např. pro Model Checking, verifikaci nebo, v této diplomové práci pro nás vhodnější, generování testovacích případů.

Většina současných SMT řešitelů podporuje SMT-LIB standardy [4], vytvořené iniciativou SMT-LIB. SMT-LIB je mezinárodní iniciativa uznávaná velkým množstvím výzkumných skupin zabývajících se výzkumem a vývojem v oblasti SMT řešitelů po celém světě. Hlavním cílem, na který se iniciativa SMT-LIB především zaměřuje, je poskytnutí jednotného standardu popisu jednotlivých teorií používaných v SMT systémech a sjednocení používaného jazyka pro vstupy a výstupy jednotlivých SMT systémů. [4] K tomu pomáhá i iniciativou SMT-LIB každoročně vypisovaná soutěž SMT řešitelů International Satisfiability Modulo Theories Competition (dále jen SMT-COMP), ve které se srovnává výkonnost jednotlivých SMT-řešitelů, a je velice atraktivní pro mnoho výzkumných skupin [4]. V současnosti je mnoho stále vyvíjených SMT řešitelů, z nichž si jako pravděpodobně nejznámější můžeme uvést SMT řešitel Z3, který je vyvíjen v laboratořích v Microsoftu. Dále si můžeme uvést například řešitelé CVC4, Yices 2, MathSAT 5 nebo SMTInterpol [14].

V posledním ročníku soutěže SMT-COMP[5], konané v červenci 2017, dopadl v celkovém srovnání napříč různými výkonnostními testy (angl. benchmark) nejlépe SMT řešitel Z3 od Microsoftu, přestože do samotné soutěže nebyl přihlášen. První místo z přihlášených SMT řešitelů obsadil SMT řešitel CVC4, následován SMT řešitel Yices 2, SMTInterpol, veriT a vampire 4.2.

3.2.1 PySMT – Python knihovna pro práci s SMT

Pro tuto práci je využito knihovny pro jazyk Python PySMT, která zprostředkovává aplikační rozhraní pro práci s různými SMT řešiteli. [1]



Obrázek 3.2: Schéma PySMT [1].

Tato knihovna umožňuje použití jakéhokoliv z následujících SMT řešitelů.

- MathSAT (http://mathsat.fbk.eu/).
- Z3 (https://github.com/Z3Prover/z3/).
- CVC4 (http://cvc4.cs.nyu.edu/web/).
- Yices 2 (http://yices.csl.sri.com/).
- CUDD (http://vlsi.colorado.edu/~fabio/CUDD/).

[5] Výsledky soutěže SMT-COMP 2017 – http://smtcomp.sourceforge.net/2017/results-toc.shtml

- PicoSAT (http://fmv.jku.at/picosat/).
- Boolector (http://fmv.jku.at/boolector/).

Pro ilustraci je zde uveden příklad řešení problému přiřazení čísel 1-9 k písmenům abecedy tak, aby platila následující rovnice
$H + E + L + L + O = W + O + R + L + D = 25$

```
from pysmt.shortcuts import Symbol, And, GE, LT, Plus, Equals, Int, get_model
from pysmt.typing import INT

hello = [Symbol(s, INT) for s in "hello"]
world = [Symbol(s, INT) for s in "world"]
letters = set(hello+world)
domains = And([And(GE(l, Int(1)),
                   LT(l, Int(10))) for l in letters])

sum_hello = Plus(hello) # n-ary operators can take lists
sum_world = Plus(world) # as arguments
problem = And(Equals(sum_hello, sum_world),
              Equals(sum_hello, Int(25)))
formula = And(domains, problem)

print("Serialization of the formula:")
print(formula)

model = get_model(formula)
if model:
  print(model)
else:
  print("No solution found")
```

Výpis 3.1: Příklad použití pySMT [1].

Což na výstup vrátí výsledek, který lze vidět na výpisu 3.2.

```
Serialization of the formula:
((((1 <= h) & (h < 10)) & ((1 <= e) & (e < 10)) & ((1 <= l) & (l < 10)) &
  ((1 <= o) & (o < 10)) & ((1 <= w) & (w < 10)) & ((1 <= r) & (r < 10)) &
  ((1 <= d) & (d < 10))) & (((h + e + l + l + o) = (w + o + r + l + d)) &
  ((h + e + l + l + o) = 25)))
o := 1
l := 3
e := 9
h := 9
d := 3
r := 9
w := 9
```

Výpis 3.2: Výstup k příkladu použití pySMT.

Na výpisu 3.2, který byl vytvořen ze vzorového příkladu použití PySMT (viz výpis 3.1), lze názorně vidět, že SMT řešitel správně přiřadil číselné konstanty k jednotlivým písmenům a tím bylo pomocí SMT řešitele dosaženo řešení zadaného problému.

Kapitola 4

Návrh nástroje pro tvorbu testovacího obsahu databáze

V této kapitole jsou v první části nejprve podrobně zpracovány požadavky na vytvářený nástroj pro generování obsahu databáze. Ve druhé části je potom samotný návrh vytvářeného nástroje, ve kterém je popsáno, jak jsou data pro testovací účely generována a případně ukládána do databáze.

4.1 Specifikace požadavků

V této podkapitole jsou strukturovaně rozebrány veškeré požadavky, které jsou na implementovaný nástroj pro generování testovacích dat kladeny, a které by měl splňovat. Nejprve jsou uvedeny základní obecné požadavky na nástroj, které lze vidět v tabulce 4.1. Tyto požadavky jsou pak případně dále v dalších podkapitolách specifikovány podrobněji.

ident.	**Popis požadavku**	**Řešeno v**
REQ1	Pro spuštění nástroje bude nutné mít správně nastavený vstupní konfigurační soubor.	4.2.1, 5.3, 6.3
REQ2	Nástroj bude generovat data, která budou sloužit pro testovací účely.	
REQ3	Nástroj bude podporovat možnost přidávat zásuvné moduly – pluginy.	
REQ4	Nástroj bude navržen tak, aby bylo možné jej ovládat přes různá rozhraní.	
REQ5	Nástroj bude mít možnost volby, kam zapisovat výstup z programu.	
REQ6	Nástroj bude umět pracovat s více systémy řízení báze dat.	

Tabulka 4.1: Tabulka specifikace obecných požadavků na nástroj dbgenx.

4.1.1 Požadavky na konfigurační soubor

V této podkapitole jsou popsány požadavky, jež jsou kladeny na vstupní konfigurační soubor. Jednotlivé požadavky se nachází v tabulce 4.2, pod kterou se pak nachází i vysvětlení k určitým pojmům v tabulce používaných.

RK1	Konfigurační soubor musí obsahovat definici struktury databáze.	
RK2	Konfigurační soubor musí obsahovat definici datových typů vytvořených uživatelem.	
RK3	Konfigurační soubor musí obsahovat definici požadovaných dat pro generování – které objekty a v jakém množství.	
RK4	Konfigurační soubor může obsahovat uživatelem zadaná sémantická omezení.	
RK5	Konfigurační soubor může obsahovat seznam názvů použitých datasetů.	
RK5	Konfigurační soubor bude ve formátu YAML.	

Tabulka 4.2: Tabulka specifikace požadavků na vstupní konfigurační soubor.

Struktura databáze – Touto položkou je míněn seznam databázových tabulek, názvy sloupců a jejich mapování na složené datové typy

Specifikace datových typů – Jedná se o definici složených datových typů, objektů, kterým budou přiřazeny jednotlivé databázové sloupce volitelně i z více tabulek. Složené datové typy se skládají ze základních datových typů, které jsou vyjmenovány a popsány níže v podkapitole 4.1.2.

Požadovaná data pro generování – Tato položka obsahuje seznam a počty jednotlivých objektů, které se budou generovat na základě datových typů nebo datasetů.

Sémantická omezení (angl. constraints) – V této položce jsou omezující podmínky, které musí generovaná data splňovat (např. hodnota < 5).

4.1.2 Požadavky na generovaná data

V této podkapitole jsou nejprve v tabulce 4.3 uvedeny požadavky na generovaná data a dále jsou pak vypsány jednotlivé základní používané datové typy, které je možno použít.

RG1	Data bude možné generovat přímo do připojené databáze.	
RG2	Data bude možné generovat jako textové SQL příkazy do výstupního souboru.	
RG3	Data se budou generovat a tisknout jako textové SQL příkazy na standardní výstup.	
RG4	Generovat data půjde na základě těchto sémantických omezení: $<, \leq, >, \geq, =, \neq$.	

Tabulka 4.3: Tabulka specifikace požadavků na generování dat.

Základní datové typy

1. Int – celočíselný datový typ o velikosti 32bitů.
2. BigInt – celočíselný datový typ o velikosti 64bitů.
3. Float – číslo s plovoucí desetinnou čárkou.
4. Boolean.
5. Date - datum ve formátu rok-měsíc-den.

6. Datetime - datum a čas ve formátu rok-měsíc-den hodina:minuta:sekunda.
7. String.

4.1.3 Datasety

Implementovaný nástroj by měl již obsahovat výběry některých předpřipravených dat – datasety. Datasety, které nástroj bude obsahovat a požadavky na ně jsou uvedeny níže v tabulce 4.4.

RD1	Nástroj bude umět generovat data náhodným výběrem z předem připravených dat – datasetů.	
RD2	Nástroj bude umět generovat data postupným výběrem položek z datasetů.	
RD3	Bude možné vytvořit si vlastní dataset.	
RD4	Nástroj bude obsahovat dataset křestních jmen.	
RD5	Nástroj bude obsahovat dataset příjmení.	
RD6	Nástroj bude obsahovat dataset měst.	
RD7	Nástroj bude obsahovat dataset adres.	
RD8	Nástroj bude obsahovat dataset států.	
RD9	Nástroj bude obsahovat dataset poštovních směrovacích čísel.	

Tabulka 4.4: Tabulka specifikace požadavků na práci s datasety.

4.1.4 Požadavky na rozhraní

V této podkapitole jsou specifikovány požadavky na rozhraní, které by měl nástroj dbgenx podporovat. Samotné požadavky jsou specifikovány v tabulce 4.5.

RI1	Nástroj bude fungovat jako samostatná konzolová aplikace.	
RI2	Nástroj bude možné ovládat v jazyce Python 3 přes volání metod tříd.	
RI3	Nástroj bude umožňovat komunikaci přes aplikační rozhraní platformy Testos.	

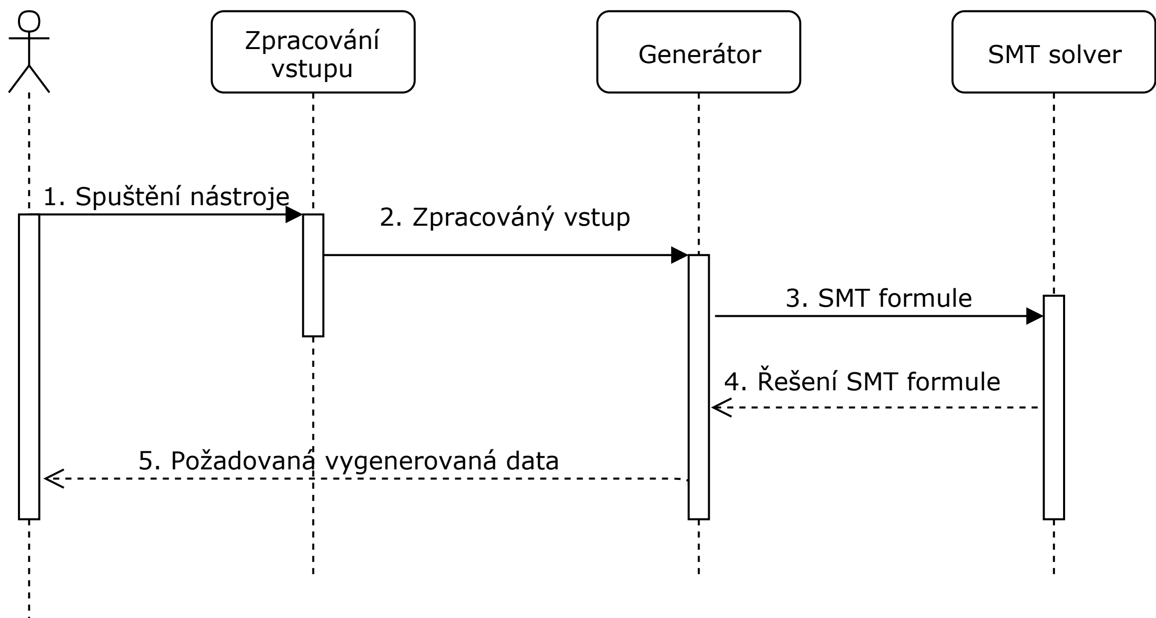
Tabulka 4.5: Tabulka specifikace rozhraní, která bude nástroj podporovat.

4.2 Návrh nástroje dbgenx

V této podkapitole je rozebrán popis činnosti vyvíjeného nástroje dbgenx. Jsou zde nejprve obecně představeny jednotlivé kroky běhu tohoto nástroje a také způsob, jakým jsou požadována testovací data, sloužící pro naplnění testovací databáze, generována. V podkapitolách jsou potom jednotlivé kroky běhu programu popsány podrobněji. Dále je v podkapitolách 4.2.6 a 4.2.7 také obsaženo představení návrhu dalších možností generování dat s využitím používaného SMT řešitele. Tyto návrhy ovšem v současném nástroji implementovány nejsou. V závěru této podkapitoly je poté popsáno aplikační rozhraní nástroje dbgenx, které bude využívat sběrnice platformy Testos – sběrnici Testos-bus.

Vyvíjený nástroj potřebuje mít na vstupu vždy zadaný, validně nastavený vstupní konfigurační soubor, na jehož základě budou poté požadovaná testovací data pro naplnění obsahu databáze generována. Tester tedy musí nejprve nastavit a specifikovat parametry pro generování požadovaných dat ve vstupním konfiguračním souboru a poté může nástroj spustit. Následně nástroj dbgenx v závislosti na požadavcích specifikovaných ve vstupním konfiguračním souboru, vygeneruje požadovaná testovací data, která uloží do určené databáze, nebo je případně zapíše ve formátu SQL příkazů pro vložení databázových dat do výstupního souboru, nebo na standardní výstup. V případě, že by z nějakého důvodu nebylo možné požadovaná data vygenerovat, například kvůli tomu, že by neexistovalo řešení, které by splňovalo uživatelem zadaná omezení, nástroj se ukončí s odpovídajícím chybovým hlášením.

Samotný proces generování dat pak funguje tak, že jakmile si nástroj načte ze vstupního konfiguračního souboru specifikaci zadání, co by se mělo generovat, předá žádost o generování dat jednomu z generátorů, který je určený pro generování dat daného datového typu. Pokud je to možné, tak generátor převede toto zadání na SMT formule, které je možno zpracovat SMT řešitelem, a nechá tento problém vyřešit SMT řešitel, který generátoru vrátí řešení zadané formule. Nakonec jsou odpovídající data poslány z generátoru na výstup nebo zapsány do databáze. Případně pokud během tohoto procesu došlo k nějaké chybě, je vypsáno chybové hlášení. Průběh generování testovacích dat je možno také vidět na obrázku 4.1.



Obrázek 4.1: Sekvenční diagram průběhu generování dat.

4.2.1 Popis formátu vstupního souboru

Vzhledem k tomu, že se jedná o rozšíření již existující bakalářské práce o využití SMT řešitele, nástroj vytvářený v této diplomové práci používá stejnou strukturu vstupního souboru, jako v předchozí bakalářské práci [17].

Pro vstupní konfigurační soubor byl zvolen serializační formát YAML [1] z důvodu jeho jednoduchého zápisu a pro člověka dobře čitelné struktury. Tento vstupní soubor se skládá až z 5 položek asociativního pole s následujícími klíči **include**, **types**, **schema**, **constraints**, **generate**. Z čehož položky **types**, **schema**, **generate** se musí ve vstupním souboru nacházet povinně, zbytek je volitelný.

types Obsahuje definice datových typů objektů, které jsou opět reprezentovány jako asociativní pole, kde klíčem je název typu a hodnotou je buďto konkrétní datový typ nebo název jiného typu z tohoto asociativního pole, nebo název některého z datasetů. Pro případ reprezentace, kdy jeden objekt obsahuje více objektu jiných typů, je možnost toto specifikovat pro {n} podtypů pomocí následujícího tvaru:

```
types:
    nazev_typu: "[nazev_podtypu]{n}"
```

Případně pro vyjádření náhodného počtu z intervalu [m, n]:

```
types:
    nazev_podtypu: "[nazev_podtypu]{m, n}"
```

schema Položka schema obsahuje opět asociativní pole, kde klíčem je název tabulky a hodnotou je další asociativní pole s dvojicemi klíč-hodnota mapující název sloupce na konkrétní datový typ z **types**.

```
schema:
    nazev_tabulky1:
        sloupec1: typ1.hodnota1
        sloupec2: typ1.hodnota2
    nazev_tabulky2:
        sloupec1: typ2.hodnota1
        sloupec2: typ1.hodnota1
```

generate Pod touto položkou je obsažen seznam textových řetězců, např.:

```
generate:
    - typ1 = 3
    - typ2 = 1
```

include Obsahuje seznam názvů datasetů, ze kterých je možné generovat zvolené hodnoty pro položky z **types**, např.:

```
include:
    - names
    - addresses
```

constraints A v této poslední položce jsou obsažena uživatelem zadaná omezení na hodnoty jednotlivých datových typů a podtypů z **types**, např.:

[1] specifikace YAML: http://yaml.org/spec/1.2/spec.html

```
constraints:
    - typ1.cislo1 > 42
    - typ1.podtyp.datum >= 2017-01-01
```

Příklad jednoduchého vstupního konfiguračního souboru, představujícího jednoduchý docházkový systém pro zaměstnance, ve kterém jsou definována všechna asociativní pole, lze vidět na výpisu 4.1. Na řádcích č. 1-3 se nachází specifikace názvů použitých datasetů. Na řádcích č. 5-13 jsou uživatelem definovány datové typy objektů employee a attendance, kdy zaměstnanci (employee) je přiřazeno 1-365 objektů docházky (attendance). Na řádcích č. 15-24 se nachází schéma databáze a mapování položek definovaných objektů ke sloupcům databázových tabulek. Řádky č. 26-27 obsahují jedno omezení, které značí, že příchod zaměstnance musí být vždy před jeho odchodem. A na závěr na řádcích č. 29-30 je specifikováno, že bude vytvořeno 10 zaměstnanců i s jejich docházkami.

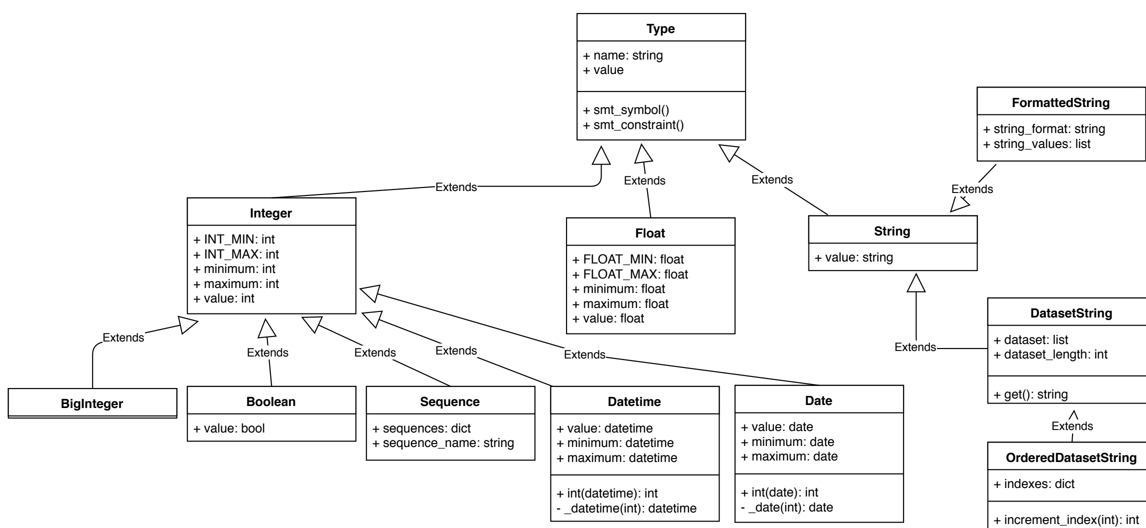
```
1  include:
2      - first_name_cs
3      - last_name_cs
4
5  types:
6      employee:
7          id: seq(0, 100000, 1)
8          first_name: first_name_cs
9          last_name: last_name_cs
10         attendance: "[attendance]"{1, 365}
11     attendance:
12         from: dinterval(2017-01-01, 2018-01-01)
13         to: dinterval(2017-01-01, 2018-01-01)
14
15 schema:
16     employees:
17         id: person.id
18         first_name: person.first_name
19         last_name: person.last_name
20
21     attendance:
22         from: employee.attendance.from
23         to: employee.attendance.to
24         employee_id: employee.id
25
26 constraints:
27     - person.attendance.from < person.attendance.to
28
29 generate:
30     - employee = 10
```

Výpis 4.1: Příklad jednoduchého vstupního konfiguračního YAML souboru.

4.2.2 Načtení dat ze vstupního souboru do vnitřní reprezentace

Po načtení vstupního souboru z disku dochází k jeho zpracování, kdy jsou zvlášť načteny do programu jednotlivé položky podle klíčů asociativního pole a uloženy do reprezentace, kterou očekává na svém vstupu část programu pro generování požadovaných hodnot.

Okamžitě dojde také ke zpracování položky **constraints**, kdy je každý jednotlivé omezení uloženo jako trojice: levý operand, operátor a pravý operand. A tyto trojice jsou dále předávány v datové struktuře seznamu. **generate** se poté načte do seznamu dvojic název typu, hodnota. Položky **schema** a **include** se předávají dále ve stejné struktuře, v jaké byly zapsány ve vstupním souboru – **schema** jako asociativní pole a **include** v seznamu. Pro uživatelem definované datové typy **types** se vytvoří pro každý jednotlivý typ vlastní objekt s unikátním názvem a tyto objekty jsou následně jako seznam předávány dále. Přehled možných tříd těchto objektů lze vidět na třídním diagramu v obrázku 4.2.



Obrázek 4.2: Diagram tříd použitých datových typů.

4.2.3 Generování dat podle zadaných omezení

Generátor dat podle uživatelem zadaných omezení generuje data s pomocí SMT řešitele, kterému jsou na vstup přivedeny jednotlivá omezení a objekty, pro které se mají hodnoty generovat. Zadaná omezení, společně s unikátními názvy objektů jsou následně převedeny do SMT formulí a následně spojeny do jedné formule. Pro tuto formuli následně SMT řešitel nalezne řešení a výsledné hodnoty jsou následně uloženy do jednotlivých objektů.

Níže je představeno, jak probíhá sestavení jednotlivých částí logické formule, která je následně předložena SMT řešiteli k nalezení řešení a vygenerování hodnot.

Tvorba SMT formulí pro jednotlivé třídy datových typů

Výsledná formule je vždy složena z jednotlivých formulí implicitních omezení na jednotlivé datové typy společně s formulemi, vytvořenými z uživatelem zadaných omezení.

Celočíselný datový typ – třídy Integer, BigInteger Typ BigInteger se od běžného celočíselného typu Integer liší pouze maximálním rozsahem, kdy Integer může obsahovat

pouze číslo velikosti 32 bitů, zatímco BigInteger až 64 bitů. Pokud pro požadovanou celočíselnou hodnotu není uživatelem specifikováno žádné omezení, tak se pro SMT řešitele pro celočíselnou proměnnou i vytvoří pouze formule v následujícím tvaru, kde *INT_MIN* je konstanta značící minimální hodnotu pro typ Integer (resp. BigInteger) a *INT_MAX* je konstanta značící maximální hodnotu pro typ Integer(resp. BigInteger).

$$INT_MIN \leq i \land i \leq INT_MAX$$

Celočíselný datový typ může být ve vstupním konfiguračním souboru také definován ve formě intervalu pomocí klíčového slova *interval(m, n)*, kde celé číslo m je spodní hranice intervalu a celé číslo n je horní hranicí intervalu. V tomto případě se pro proměnnou i předá SMT řešiteli následující formule.

$$m \leq i \land i \leq n$$

V případě, že by pro zadanou proměnnou bylo uživatelem ve vstupním konfiguračním souboru specifikováno i nějaké omezení v **constraints**, přidalo by se toto omezení k formulím výše. Formule pro omezení $i < 5$ by poté vypadala následovně.

$$(INT_MIN \leq i \land i \leq INT_MAX) \land i < 5$$

Ke generování typů Integer a BigInteger je v SMT řešiteli využito celočíselné aritmetiky.

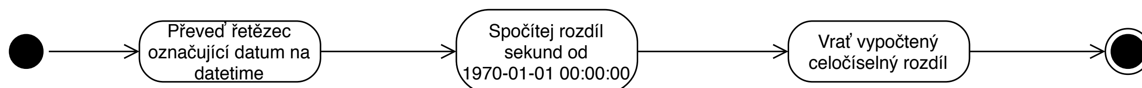
Boolovský datový typ – třída Boolean Boolovský datový typ může nabývat pouze dvou hodnot – pravda (angl. true) a nepravda (angl. false). Přestože SMT řešitelé jsou rozšířením SAT řešitelů, které pracují pouze s Boolovskými hodnotami, v nástroji dbgenx se převádí před vytvořením samotné formule boolovské hodnoty na typ Integer, nabývající hodnot 0 pro nepravdu a 1 pro pravdu. Je to z důvodu jednotnější práce s SMT řešitelem v rámci vyvíjeného nástroje. Pro Boolovský datový typ je tedy v SMT řešiteli využita také aritmetika celých čísel.

Sekvence – třída Sequence Hodnoty pro objekty třídy Sequence se generují na základě třídního sekvenčního čítače. který se každou novou hodnotou sekvence inkrementuje o daný krok. SMT řešiteli se tedy předává pouze jednoduchá formule $i = hodnota_tae$. Přestože se na první pohled může zdát, že pro generování sekvencí nemá použití SMT řešitele význam, není tomu úplně tak. Stále je totiž možné mít ve vstupním konfiguračním souboru v **constraints** některé omezení, ve kterém se bude vyskytovat hodnota typu sekvence, a tudíž i zde má SMT řešitel svůj význam. Jelikož se u sekvencí jedná o celočíselný datový typ, SMT řešitel zde opět využívá celočíselnou aritmetiku.

Typ kalendářního data a času – třída Datetime Hodnoty datové typu kalendářního data a času nejsou typem, který by byl podporován SMT řešitelem, a proto se generují tak, že se nejprve tento typ převede na celočíselný datový typ a následně se vytvoří SMT formule shodně jako u celočíselného datového typu popsaného výše. Typ kalendářního data a času může být také ve vstupním konfiguračním souboru definovaný ve formě intervalu, kdy je použito klíčového slova *dinterval(datum1, datum2)*, kde proměnná *datum1* ve formátu YYYY-MM-DD HH:mm:SS spodní hranicí intervalu a proměnná *datum2*, která je zapsána ve stejném formátu jako *datum1*, značí horní hranici intervalu. V SMT řešiteli se využívá pro generování hodnot datového typu kalendářního data a času celočíselné aritmetiky.

Samotný převod hodnot typu kalendářního data a času na celočíselný datový typ lze vidět na diagramu aktivit na obrázku 4.3. Převod je prováděn tak, že je nejprve pevně určen

výchozí časový bod s časovým razítkem 1970-1-1 00:00:00, kterému je přiřazena celočíselná hodnota 0. Následně je vypočten rozdíl, kolik sekund je mezi stanoveným výchozím časem a časem který je potřeba převézt na celočíselný datový typ. Takto vypočtený rozdíl sekund je potom výslednou hodnotou, která může být předložena SMT řešiteli.



Obrázek 4.3: Diagram aktivit převodu kalendářního data a času na celočíselnou hodnotu.

Typ kalendářního data – třída Date Typ kalendářního data je potřeba, stejně jako typ kalendářního data a času, také kvůli řešení zadaných omezení pomocí SMT řešitele převést na celočíselný datový typ. Jediný rozdíl oproti převodu z typu kalendářního data a času na celočíselný datový typ je v tom, že pro typ kalendářního data se převádí na celočíselné hodnoty počet dnů, na rozdíl od rozdílu počtu sekund.

Typ s plovoucí desetinnou čárkou – Float Pro typ s plovoucí desetinnou čárkou je formule SMT řešitele generována naprosto stejně, jako v případě celočíselných typů. Rozdíl je pouze v minimální a maximální hodnotě možné generované hodnotě. Typ s plovoucí desetinnou čárkou je také možné definovat jako interval, pomocí klíčového slova *finterval(m, n)*, kde m je desetinné číslo označující spodní hranici intervalu a n je desetinné číslo značící horní hranici intervalu. Pro generování typu Float je využito aritmetiky reálných čísel.

Opakované generování objektů

Vzhledem k tomu, že se jedná data určena převážně pro testovací účely, tak je vyžadováno, aby se jednotlivé generované objekty od sebe navzájem lišily a tím bylo dosaženo co nejvyššího pokrytí možných kombinací testovacích dat. Jelikož SMT řešitelé mohou být deterministické, je potřeba s každým dalším generovaným objektem upravit vstupní formuli SMT řešitele, tak aby bylo nalezeno jiné, ještě nevyužité řešení problému zadaných omezení.

Na výpisu 4.3 lze vidět příklad sestavené SMT formule pro objekt, obsahující dvě hodnoty a a b, který je specifikován vstupním konfiguračním souborem z výpisu 4.2. A na výpisu 4.4 je následně vidět sestavená SMT formule pro druhý generovaný objekt, definovaný ve vstupním konfiguračním souboru z výpisu 4.2. Na prvních třech řádcích obou formulí ve výpisu 4.3 a ve výpisu 4.4 jsou prakticky shodné formule, zatímco ve výpisu 4.4 stojí za povšimnutí řádek č. 4, který se ve výpisu 4.3 nenachází. Na tomto řádku č. 4 se vyskytuje část formule zaručující, že se nově generovaný objekt bude lišit od předchozího objektu. Uvedený výpis 4.4 se vztahuje k případu, kdy z nalezeného řešení pro první formuli z výpisu 4.3 byly vygenerovány následující hodnoty:

$$object.a = 0$$

$$object.b = 0$$

```
types:
    object:
        a: int
        b: int

schema:
    table:
        a: object.a
        b: object.b

generate: ["object = 2"]

constraints:
    - object.a <= 42
```

Výpis 4.2: Vstupní soubor k příkladu generování různých hodnot.

```
((((-2147483648 <= object.a) & (object.a <= 2147483647)) &
  (object.a <= 42)) &
 ((-2147483648 <= object.b) & (object.b <= 2147483647)))
```

Výpis 4.3: Příklad SMT formule prvního objektu.

```
(((((-2147483648 <= object.a) & (object.a <= 2147483647)) &
   (object.a <= 42)) &
  ((-2147483648 <= object.b) & (object.b <= 2147483647))) &
 (! ((object.a = 0) & (object.b = 0))))
```

Výpis 4.4: Příklad SMT formule pro druhý objekt.

4.2.4 Generování dat pro řetězce

Pro generování řetězců jsou použity dvě různé metody. První možnou metodou je generování náhodného alfanumerického řetězce. Druhou možnou metodou je generování řetězce, který je poskládán z již vygenerovaných hodnot objektů nebo hodnot některého z datasetů. Tento řetězec je dále nazýván jako "formátovaný řetězec".

4.2.5 Generování dat z datasetů

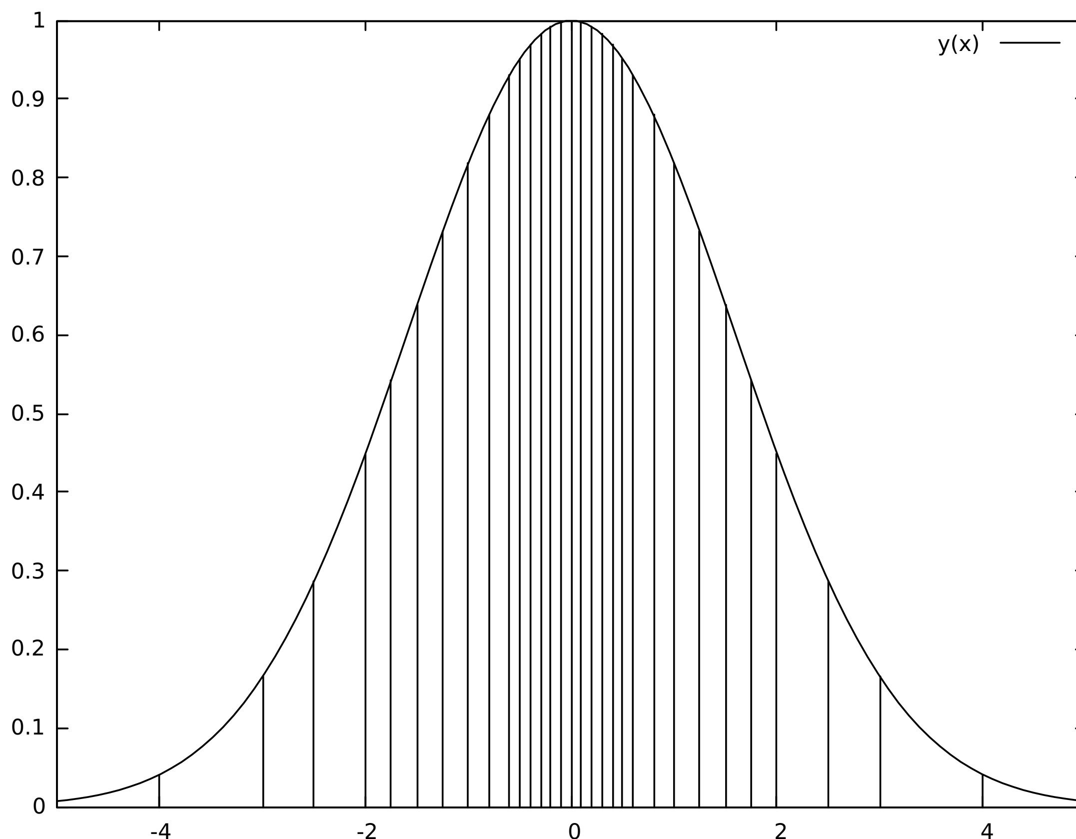
Generování hodnot z datasetů probíhá jednoduchým náhodným výběrem hodnot z daného datasetu. Ve speciálním případě, kdy je prvním prvkem datasetu klíčové slovo "ORDERED_DATASET", tak se jedná o uspořádaný dataset. Výběr hodnot z takového datasetu probíhá sekvenčně. V případě využití všech hodnot uspořádaného datasetu začíná výběr znova od prvního prvku daného datasetu.

4.2.6 Generování dat podle zadaného rozložení pravděpodobnosti

Jako jedno z možných rozšíření pro generátor testovacích dat nástroje dbgenx, které je momentálně pouze ve fázi návrhu, by bylo přidání možnosti generování vybraných testo-

vacích dat podle uživatelem zadané funkce hustoty pravděpodobnosti. Doména hodnot by byla rozdělena do několika intervalů, podle očekávaného počtu generovaných prvků, kdy jeden interval by byl přiřazen právě jednomu generovanému prvku. Intervaly v okolí více pravděpodobných hodnot by byly poměrně menší k intervalům z okolí hodnot s nižší pravděpodobností výskytu, jak lze vidět na ilustračním obrázku 4.4.

Potom, co bude vstupní doména rozdělena mezi jednotlivé intervaly, je možné tyto intervaly převést na formule a předkládat SMT řešiteli, který nám vygeneruje a vrátí výsledné požadované hodnoty pro účely testování.



Obrázek 4.4: Rozdělení na intervaly podle funkce hustoty pravděpodobnosti.

4.2.7 Generování podle pokrytí vstupních domén

Jedná se o další případné rozšíření nástroje dbgenx o možnost generování na základě pokrytí zadané vstupní domény, které je také pouze ve fázi návrhu, a u kterého se s implementací čeká na společnou sběrnici platformy Testos – Testos-bus. K tomuto rozšíření je nejprve potřeba, aby uživatel určil rozdělení zvolené vstupní domény na jednotlivé bloky. V případě číselných datových typů je možné vstupní doménu rozdělit do bloků podle intervalů, a tyto intervaly převedené do formulí předkládat SMT řešiteli. V případě řetězců lze vstupní doménu rozdělit například podle jednotlivých datasetů (např. doména křestních jmen může být rozdělena na ženská a mužská jména). Toto rozšíření nástroje dbgenx by bylo vhodné také propojit s kombinačním generátorem platformy Testos [16]. Bohužel v době vzniku této práce ještě nebyla sběrnice platformy Testos – Testos-bus implementována.

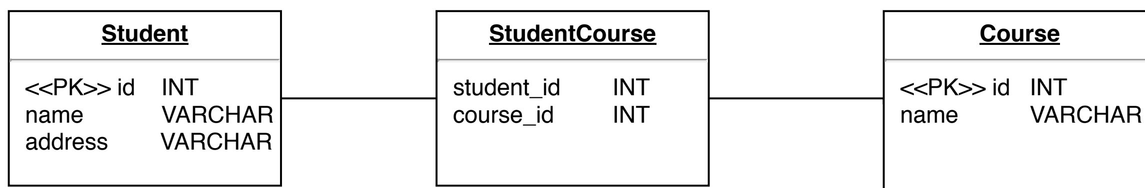
4.2.8 Vkládání dat do databáze

Poté co jsou vygenerovány hodnoty všech objektů přes generátor využívající SMT řešitel a generátor řetězců, může proběhnout jejich namapování a vložení do databáze. Aby odpovídaly cizí klíče odkazující na další databázové tabulky, kdy jednotlivé atributy generovaného objektu mohou být obsaženy ve více databázových tabulkách, je potřeba aby byly data vkládány do databázových tabulek ve správném pořadí. Pro určení pořadí, do které tabulky se budou data vkládat první se vygenerované hodnoty objektů nejprve roztřídí podle úrovně strukturálního zanoření – zjednodušeně podle počtu teček ve svém názvu. Pak je potřeba jednotlivé proměnné objektů namapovat k jednotlivým sloupcům databázových tabulek, definovaných v položce **schema** ve vstupním konfiguračním souboru. To probíhá iterativně tak, že se vezmou nejprve všechny prvky se zanořením 1 a zkusí se namapovat na sloupce v tabulkách, pak se vezmou prvky se zanořením 2 a zkouší se namapovat na sloupce databázových tabulek, a takto to pokračuje až do okamžiku, dokud nejsou všechny hodnoty namapovány na jednotlivé sloupce databázových tabulek. Ve chvíli, kdy je možno při iterativním mapování přiřadit ke všem sloupcům databázové tabulky vygenerované hodnoty, dojde k vytvoření SQL příkazu INSERT pro danou tabulku. Jakmile se vygenerují všechny SQL příkazy pro všechny databázové tabulky, může dojít k jejich složení do databázové transakce a následně mohou být vloženy do databáze, nebo případně být zapsány do výstupního souboru, nebo vytištěny na standardní výstup.

Pro možnost provázání jednotlivých objektů mezi sebou v databázových tabulkách, kdy by ani jeden objekt byl z hlediska jeho struktury obsahem druhého objektu, je umožněno pomocí konstrukce klíčového slova **FROM** a odkazu na konkrétní datový typ některého objektu. Toho je možné využít například pokud je potřeba propojit tabulky, které mezi sebou mají relaci typu M:N. Příklad použití klíčového slova **FROM** pro generování obsahu tabulek v relaci M:N lze názorně vidět ve výpisu 4.5, který by byl použit pro databázové schéma, jež lze vidět na obrázku 4.5.

```
1 schema:
2     :
3         a: object.a
4         b: object.b
5
6 generate: ["object = 2"]
7
8 constraints:
9     - object.a <= 42
```

Výpis 4.5: Část vstupního konfiguračního souboru s položkou schema pro demonstraci použití FROM.



Obrázek 4.5: Schéma databáze pro ukázku použití ke generování tabulek v M:N relaci.

V případě, že je využito klíčového slova **FROM**, jsou nejprve vygenerovány všechny objekty, jejich hodnoty jsou prvně namapovány na ostatní sloupce, a pak je sloupci s klíčovým slovem **FROM** náhodně přiřazena některá z odpovídajících vygenerovaných hodnot.

4.2.9 Návrh aplikačního rozhraní pro komunikaci s dalšími nástroji platformy Testos

Pro komunikaci s dalšími nástroji platformy Testos bude využito společné sběrnice Testos-bus, která ještě ale nebyla dokončena, jak již bylo v této diplomové práci zmíněno dříve. Až bude společná sběrnice zprovozněna a ostatní nástroje platformy Testos ji v sobě budou mít implementovány, mohl by nástroj dbgenx například získávat informace, která data dále generovat k pokrytí vstupní domény testovacími daty, díky informacím získaným z kombinačního generátoru [16].

Sběrnice Testos-bus by měla umožňovat zasílat následující tři typy zpráv. [5]

- **Signal** – Jedná se o typ zprávy, který neočekává žádnou odpověď.
- **Request** – Jedná se o typ zprávy, kdy odesílatel očekává od příjemce odpověď.
- **Response** – Jedná se o odpověď na zprávu typu *Request*.

Každá zpráva se bude povinně skládat z následujících dvou částí. [5]

1. Název typu zprávy – Formát názvu typu zprávy není pevně definován, ale pro možnost jednoznačné identifikace zpráv mezi jednotlivými komponentami platformy Testos by měla každá zpráva začínat řetězcem **org.testos.nazev_aplikace**.
2. Obsah zprávy (dále payload) – Data, která se pomocí zprávy předávají. Preferovaným serializačním formátem v platformě Testos je formát JSON[2].

Aplikační rozhraní nástroje dbgenx bude umožňovat přijímat pouze typy zasílaných zpráv, které počítají se schématem dotaz-odpověď – budou přijímat pouze typ zprávy *Request* a odpovídat typem zprávy *Response*. Ač po sběrnici platformy mohou chodit zprávy jakéhokoliv typu, nástroj dbgenx bude přes své aplikační rozhraní reagovat pouze na tři druhy zpráv, které jsou podrobněji rozebrány v podkapitolách níže.

org.testos.dbgenx.generate

Při zprávě typu org.testos.dbgenx.generate nástroj dbgenx očekává, že mu v části payload přijde na vstup objekt ve formátu json, který je strukturovaný podobně jako vstupní konfigurační soubor ve formátu YAML. Tento objekt bude opět obsahovat položky **include**, **types**, **schema**, **constraints** a **generate**. S tím, že jediný rozdíl oproti struktuře vstupního konfiguračního souboru bude, že položky seznamu **constraints** již budou předávány jako trojice a položky seznamu **generate** jako dvojice, oproti reprezentaci v seznamech řetězců, jako tomu bylo u vstupního konfiguračního souboru. Příklady vstupů a zmíněné rozdíly ve struktuře vstupů lze vidět na výpisech 4.6 a 4.7.

[2]serializační formát JSON – `https://www.json.org`

```
1  includes:
2      - names
3  types:
4      object:
5          a: int
6          b: int
7  schema:
8      tnumbers:
9          num1: object.a
10         num2: object.b
11 generate:
12     - "object = 2"
13 constraints:
14     - object.a <= 42
```

Výpis 4.6: Ukázka vstupního konfiguračního souboru pro srovnání se vstupem ve formátu JSON.

```
1  {
2      "includes": ["names"],
3      "types": {
4          "object": {
5              "a": "int"
6              "b": "int"
7          }
8      },
9      "schema": {
10         "tnumbers": {
11             "num1": "object.a"
12             "num2": "object.b"
13         }
14     },
15     "generate": [
16         ["object", 2]
17     ],
18     "constraints": [
19         ["object.a", "<=", 42]
20     ]
21 }
```

Výpis 4.7: Ukázka JSON vstupu přes aplikační rozhraní pro typ zprávy org.testos.dbgenx.generate.

Odpověď na toto volání ve své části payload obsahuje pole vygenerovaných objektů s jejich hodnotami ve formátu json, viz. výpis 4.8.

```
[
    {
        "object.a": 1,
        "object.b": 2
    },
    {
        "object.a": 3,
        "object.b": 4
    }
]
```

Výpis 4.8: Ukázka JSON odpovědi aplikačního rozhraní na typ zprávy org.testos.dbgenx.generate.

org.testos.dbgenx.generate_sql

Při tomto volání této zprávy nástroj dbgenx očekává stejný obsah pro payload, jako při volání **org.testos.dbgenx.generate**, ale na rozdíl od předchozího volání ve své odpovědi vrací seznam řetězců SQL příkazů (viz výpis 4.9).

```
[
    "INSERT INTO tnumbers (num1, num2) VALUES (1, 2);",
    "INSERT INTO tnumbers (num1, num2) VALUES (3, 4);"
]
```

Výpis 4.9: Ukázka odpovědi org.testos.dbgenx.generate_sql.

org.testos.dbgenx.datasets

Toto volání očekává v části payload pouze seznam názvů datasetů, které samotný nástroj dbgenx obsahuje (jsou uloženy v adresáři datasets), ve formátu json (viz. výpis 4.10. Jako odpověď vrací json objekt s obsahem požadovaných datasetů, jak lze vidět na výpisu 4.11.

```
["names", "cities", "countries", "currencies"]
```

Výpis 4.10: Ukázka příchozí zprávy org.testos.dbgenx.datasets.

```
{
    "names": ["Abadon", "Abigail", "Abdon", "Adam", ...],
    "cities": ["Praha", "Brno", "Ostrava", "Olomouc", ...],
    "countries": ["Angola", "Benin", "Maroko", ...],
    "currencies": ["CZK", "USD", "EUR", "GBP", ...]
}
```

Výpis 4.11: Ukázka odpovědi org.testos.dbgenx.datasets.

Kapitola 5

Implementace nástroje dbgenx

V této kapitole se nachází popis toho, jak byl nástroj dbgenx implementován. Nejprve jsou představeny možnosti jeho ovládání, následované krátkou zmínkou o použitých knihovnách. Následně jsou popsané některé významné implementační detaily samotného nástroje.

5.1 Spuštění nástroje dbgenx

Po nainstalování nástroje a všech potřebných závislostí, především SMT řešitele, je možné nástroj spustit jako konzolovou aplikaci následujícím příkazem.

```
dbgenx <nazev_vstupniho_souboru> [parametry]
```

Kde názvem vstupního souboru je cesta ke konfiguračnímu souboru, popsanému v podkapitole 4.2.1. A možné parametry pro spuštění jsou následující:

- **-h, --help** – Parametr pro vypsání nápovědy.
- **--host** – Parametr pro adresu serveru s databází (hostname).
- **--port** – Parametr pro číslo portu.
- **-u, --user** – Parametr pro jméno uživatele databáze.
- **-p, --password** – Parametr pro heslo uživatele databáze.
- **-d, --database** – Parametr pro název databáze.
- **-s, --dbms** – Parametr pro typ databázového systému, volby jsou mysql, postgres, sqlite, ms-sql
- **-o, --output** – Parametr pro název výstupního souboru, do kterého jsou zapsány výsledné SQL příkazy.
- **--dataset** – Parametr obsahující cesta k adresáři obsahující datasety, jsou-li potřeba. Implicitně je nastavena cesta k aktuálnímu pracovnímu adresáři.

V případě, že nejsou vyplněny všechny potřebné parametry pro připojení k databázi, vygenerované SQL INSERT příkazy se zapíší do souboru specifikovaném parametrem **-o**. Pokud není zadán ani výstupní soubor, SQL příkazy jsou vypsány na standardní výstup.

5.1.1 Další možnosti použití nástroje dbgenx

kdy po nainstalování balíčku **dbgenx** (a potřebných závislostí) lze k použití nástroje z jakéhokoliv zdrojového souboru, napsaného v jazyce Python, nástroj **dbgenx** použít pomocí příkazu *import dbgenx*.

Po dokončení společné sběrnice pro všechny nástroje platformy Testos, bude s nástrojem také možné komunikovat a používat jej přes aplikační rozhraní sběrnice Testos-Bus, kdy část aplikačního rozhraní, nacházejícího se v nástroji dbgenx, byla již prototypově implementována a čeká na samotné napojení, až bude sběrnice Testos-Bus v provozu.

5.2 Použité knihovny a nástroje pro implementaci nástroje dbgenx

Vyvíjený nástroj pro tvorbu obsahu databáze pro účely testování software je celý implementován v jazyce Python, konkrétně ve verzi 3.6. Kromě využití standardních knihoven, jež jsou obsaženy v samotném jazyce Python, jsou použity i následující externí knihovny:

- **pyYAML**[1] – Tato knihovna je použita pro zpracování vstupního konfiguračního souboru a uložení načtených dat do vnitřních datových struktur nástroje.
- **pySMT** – Knihovna pySMT byla již představena v podkapitole 3.2.1, je použita jako prostředek pro použití SMT řešitele ke generování dat.
- **sqlalchemy**[2] – Jedná se o knihovnu, která slouží pro práci s databázemi a umožňuje tak ukládání generovaných dat do cílové testovací databáze.

5.3 Implementační detaily zpracování konfiguračního vstupního souboru

Vstupní konfigurační soubor je zpracován v modulu **cli.py** ve funkci *parse_input(filename)*. Po zavolání *parse_input(filename)* s parametrem vstupního konfiguračního souboru, je z této funkce volána funkce *parse_yaml(filename)*, která pomocí knihovního volání funkce *yaml.safe_load(filestream)* načte data ze vstupního konfiguračního souboru do základních datových struktur jazyka Python – Do seznamů a slovníků (asociativních polí). Data z položek **generate** a **constraints** jsou ještě funkcemi *get_generates()* a *get_constraints()* předzpracovány a převedeny ze seznamů řetězců na seznam n-tic. Takto načtená data ze vstupního konfiguračního souboru jsou následně předána ke generování, pomocí funkce *generate_data()* jsou pak dále předána ke generování.

5.4 Implementační detaily generování dat podle zadaných omezení

Proces generování požadovaných testovacích dat je řešen převážně v modulu **generator.py**, kde se nachází definice tříd *GeneratorSMT* a *Generator* a stěžejní funkce *generate_data()*, která se stará o celý proces generování testovacích dat podle uživatelem zadaných omezení.

[1] pyYAML – http://pyyaml.org/wiki/PyYAML

[2] sqlalchemy – https://www.sqlalchemy.org/

Funkce *generate_data()* dostává na vstup načtená data ze vstupního konfiguračního souboru, které byly předzpracovány v předchozí fázi, jež je popsána v podkapitole 5.3, a také instanci objektu do kterého se vygenerovaná testovací data budou vkládat. Což případně umožní generovat výstup i jinak, než ve formátu pro jazyk SQL, pokud by to v budoucnu bylo potřeba rozšířit o další formát výstupu.

Ve funkci *generate_data()* se nejprve z uživatelsky zadaných typů vytvoří objekty datových typů, které lze vidět na diagramu 4.2. Pak se vytvoří instance třídy *GeneratorSMT*, která si následně vytvoří pomocí své metody *GeneratorSMT.smt_constraints()* seznam SMT formulí, které poté po zavolání metody *GeneratorSMT.solution_generator()* spojí do jediné formule, kterou nechá vyřešit SMT řešitele.

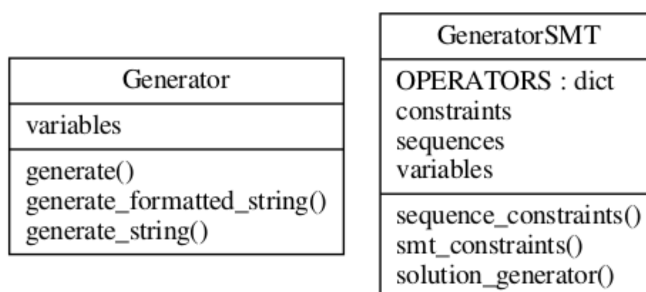
Metoda *GeneratorSMT.solution_generator()* je implementována jako metoda generátoru jazyka Python a tudíž využívá líné evaluace (angl. lazy evaluation), kdy se další požadovaná hodnota generuje pomocí SMT řešitele, až když je to potřeba. Což může mít při velkém množství generovaných hodnot mít přínos v šetření použité paměti nástroje. Tato metoda také zaručuje, že každý další vygenerovaný objekt se bude od předchozího vygenerovaného objektu lišit.

Ukázka této metody se nachází ve výpisu 5.1, kde se na řádcích č. 11-27 v cyklu generuje vždy nový model řešení, které je nalezeno SMT řešitelem na řádku č. 12, které je z řádku č. 15 navráceno o úroveň výše. Na řádcích č. 18-25 je potom vytvořena SMT formule, odpovídající aktuálně vygenerovanému řešení. Tato formule je na řádku č. 27 znegována a přidána k již používaným omezením, což znamená, že žádné další řešení nemůže být stejné, jako již některé jednou vygenerované.

```
1  def solution_generator(self, smt_constraints):
2      """Solution generator that yields always different solution
3
4      Args:
5          smt_constraints (lst): list of SMT formulas
6
7      Returns solution of SMT formulas as SMT model.
8
9      """
10     seq_names = [seq.name for seq in self.sequences]
11     while True:
12         model = self._get_solution(smt_constraints)
13         if not model:
14             break
15         yield model
16
17         # next time find another solution
18         to_add = None
19         for i, (symbol, val) in enumerate(model):
20             if symbol.symbol_name().split('#')[0] in seq_names:
21                 continue # sequences has own rules
22             if not to_add:
23                 to_add = Equals(symbol, val)
24             else:
25                 to_add = And(Equals(symbol, val), to_add)
26         if to_add:
27             smt_constraints.append(Not(to_add))
```

Výpis 5.1: Generátor řešení podle zadaných omezení.

Poté co jsou vygenerovány hodnoty objektů jednotlivých datových typů, které bylo možno vygenerovat z SMT formulí přes objekt třídy *GeneratorSMT* s pomocí SMT řešitele, přijde na řadu generování hodnot pro datové typy, které třída *GeneratorSMT*, neumožňuje generovat – konkrétně se jedná o dogenerování hodnot objektů třídy *String*. Tyto hodnoty řetězců jsou poté vygenerovány pomocí třídy *Generator* a některé z jejich metod. Metody, které mají jak třída *Generator* tak třída *GeneratorSMT* lze vidět na diagramu tříd na obrázku 5.1.



Obrázek 5.1: Diagram tříd Generator a GeneratorSMT.

A na závěr, když už jsou vygenerovány všechny hodnoty pro všechny objekty jednotlivých datových typů, dojde k jejich předání dále do objektu třídy pro generování SQL.

5.5 Implementační detaily převodu dat na SQL a jeho vkládání do Databáze

Vkládání dat do SQL databází probíhá v modulu **sql.py**. Jako první ze všeho jsou jednotlivé objekty s vygenerovanými hodnotami mapovány podle jmen objektů k databázovým tabulkám v metodě *SQL.variables_to_sql()*. Pak proběhne ještě náhodné přiřazení odpovídajících objektů ke sloupcům tabulek, které obsahují klíčové slovo FROM, jehož použití bylo představeno v podkapitole 4.2.8. Ve chvíli kdy máme všechny sloupce všech tabulek přiřazené k sobě navzájem, tak proběhne vytvoření textových řetězců příkazů *SQL INSERT INTO*. Tyto řetězce jsou pak přidány do seznamu SQL příkazů. Na závěr proběhne zápis vytvořených SQL příkazů do souboru při použití metody *SQL.write_file()*, nebo vytisknutím na standardní výstup, pokud dojde k volání metody *SQL.stdout()*. Nebo ještě případně může dojít dojít k provedení těchto SQL příkazů v databázi. K tomu je možné použít metodu *SQL.db_insert()*, která se nejprve k databázi připojí a následně SQL příkazy spustí v jedné transakci.

Kapitola 6

Ověření kvality nástroje dbgenx

Testování a ověření funkčnosti vytvořeného nástroje bylo prováděno na různých úrovních abstrakce. Nejprve byly vytvořeny testy na nejnižší úrovni – jednotkové testy. Dále proběhlo ověření funkčnosti vybraných vzájemných závislostí mezi některými komponentami vytvářeného nástroje pomocí integračního testování. A na závěr byly vytvořeny testy podle specifikace požadavků definované v podkapitole 4.1 a ověření funkčnosti nástroje dbgenx jako celku, jeho spouštěním na předpřipravených vstupních souborech.

Pro ověření funkčnosti a účely demonstrace přínosu implementace nástroje dbgenx bylo provedeno i krátké srovnání s nástrojem db-gen, na který tato diplomová práce částečně navazuje.

6.1 Jednotkové testování

Jednotkové testy testují pouze malé části kódu, většinou na úrovni funkcí a metod, který je odstíněn od vnějších závislostí. Spouštění jednotkových testů tedy probíhá velmi rychle.

Pro jednotkové testování bylo využito vestavěných knihoven Pythonu, konkrétně knihoven *unittest* a *mock*. Při psaní jednotkových testů bylo usilováno o co největší pokrytí všech řádků kódů, aby bylo možné kdykoliv a rychle ověřit, že všechny řádky zdrojového kódu všech metod a funkcí implementovaného nástroje jsou proveditelné, a že je místo ve zdrojovém kódu, kde se případné chyby nachází, jednoduše dohledatelné a opravitelné.

Pro měření metriky pokrytí všech řádků kódu bylo využito externí knihovny *coverage.py*.

Samotné 100 % pokrytí všech řádků ale nemusí vždy dávat smysl. Jednak má jen částečnou vypovídající hodnotu o kvalitě samotných testů, a pak protože pokrytí jednotkovými testy pro některé části zdrojového kódu nemusí dávat smysl. Konkrétně například u funkce *parse_yaml()*, která pouze otevře soubor pro čtení a pro načtení použije knihovní funkci *yaml.safe_load()*, by po odstranění externích závislostí (nahrazením mock objektem) by vlastně neobsahovala žádný testovatelný kód, kromě toho nahrazeného mockem.

Vytvořené jednotkové testy jsou rozděleny po modulech, podle toho který konkrétní modul testují. Pro spuštění všech jednotkových testů byl vytvořen testovací skript, který lze spustit pomocí následujícího volání:

```
python3 tests/unit/test_runner.py
```

6.2 Integrační testování

Pro testovaní na vyšší úrovni, než jsou jednotkové testy je využíváno testů integračních. Ty se zaměřují na ověření funkčnosti větších celků a jejich vzájemné komunikace mezi sebou nebo s dalšími externími komponentami (např. zápis na disk, komunikace s databází). Integrační testování bylo prováděno také s pomocí knihovny *unittest*.

6.3 Akceptační testování

Pro testy na nejvyšší úrovni byly vytvořeny automatizované akceptační testy, jejímíž testovacími scénáři je pokryta část požadavků definovaných v podkapitole 4.1. Níže lze vidět v tabulce 6.1, jaké byly vytvořeny testovací scénáře pro ověření požadavků týkajících se vstupního konfiguračního souboru.

Testovací scénář	**Očekávaný výsledek**	**Požadavek**
Vstupní konfigurační soubor není zadán	Program skončí s chybovým hlášením	REQ1
Vstupní konfiguračnímu soubor neobsahuje položku struktury databáze	Program skončí vyjímkou	RK1
Vstupní konfiguračnímu soubor neobsahuje položku specifikací datových typů	Program skončí vyjímkou	RK2
Vstupní konfiguračnímu soubor neobsahuje položku, která data se mají generovat	Program může dále běžet	RK3
Vstupní konfiguračnímu soubor neobsahuje položku s omezeními na data	Program může dále běžet	RK4
Vstupní konfiguračnímu soubor neobsahuje položku datasetů	Program může dále běžet	RK5
Vstupní konfiguračnímu soubor není ve správném formátu	Program skončí s výjimkou	RK6

Tabulka 6.1: Tabulka akceptačního testování pro správnost zadání vstupního konfiguračního souboru.

K akceptačnímu testování bylo využíváno BDD frameworku *behave*[1], který částečně umožňuje psaní testů v přirozeném jazyce. Kdy se do souborů, nazývajících se features se zapisují testy pomocí konstrukcí "Given, When, Then", jak bylo popsáno v podkapitole 2.4. A pak do souborů nazývajících se steps jsou testy k jednotlivým položkám ze souborů features implementovány. Což si můžeme ukázat na příkladu z tutoriálu BDD frameworku, kde na výpisu 6.1, kde je vidět, jak vypadají feature soubory a na výpisu 6.2, kde lze vidět, jak v souboru step vypadá programová implementace položek ze souboru feature.

[1] BDD framework Behave – https://behave.readthedocs.io/en/latest/

```
Feature: showing off behave

  Scenario: run a simple test
     Given we have behave installed
      When we implement a test
      Then behave will test it for us!
```

Výpis 6.1: Příklad feature souboru (zdroj: stránky frameworku BDD.)

```
from behave import *

@given('we have behave installed')
def step_impl(context):
    pass

@when('we implement a test')
def step_impl(context):
    assert True is not False

@then('behave will test it for us!')
def step_impl(context):
    assert context.failed is False
Feature: showing off behave
```

Výpis 6.2: Příklad step souboru (zdroj: stránky frameworku BDD.)

6.4 Porovnání nástroje dbgenx s předchozím nástrojem db-gen

V této podkapitole se nachází srovnání nového nástroje dbgenx, vyvíjeného v rámci této diplomové práce, se starším nástrojem db-gen, který byl před dvěmi lety vytvořen v rámci cizí bakalářské práce.

Nově vytvořený nástroj dbgenx využívá ke své práci SMT řešitel a měl by díky tomu lépe zvládnout generovat data, která jsou závislá na zadaných omezeních. Zatímco starší nástroj db-gen, který je založen na pseudonáhodném generátoru, kdy se pro zadaná omezení snaží nejprve data vygenerovat a pak ověřuje, zda zadaná omezení splnil. To, že si nový nástroj dbgenx s SMT řešitelem vede v této disciplíně o poznání lépe lze vidět na příkladu, kdy se jednotlivé nástroje snaží najít řešení pro jednoduchý vstupní soubor níže na výpisu 6.3. Pro tento příklad totiž nástroj db-gen nebyl ani při 50-krát opakovaném spuštění nalézt řešení pro oba generované objekty zároveň. Naproti tomu nástroj dbgenx řešení bez problému nalezl. Toto získané řešení můžeme vidět na výpisu 6.4.

```
types:
    number:
        a: interval(0,1000000)
        b: interval(0,1000000)
        c: interval(0,1000000)
        d: interval(0,1000000)
        e: interval(0,1000000)

schema:
    tnumber:
        a: number.a
        b: number.b
        c: number.c
        d: number.d
        e: number.e

constraints:
    - number.a > number.b
    - number.b < number.c
    - number.c < number.d
    - number.d = 10
    - number.a > 20
    - number.b = number.e

generate:
    - number = 2
```

Výpis 6.3: Vstupní konfigurační soubor, pro který starý nástroj db-gen nebyl schopen nalézt řešení.

```
INSERT INTO tnumber (a, b, c, d, e) VALUES ('21', '0', '1', '10', '0');
INSERT INTO tnumber (a, b, c, d, e) VALUES ('22', '1', '2', '10', '1');
```

Výpis 6.4: Řešení nalezené nástrojem dbgenx.

Při srovnávání obou nástrojů byly zároveň objeveny některé chyby nástroje db-gen, které budou reportovány v repozitáři nástroje db-gen[2].

[2]https://pajda.fit.vutbr.cz/testos/db-gen/issues

Kapitola 7

Závěr

V této práci byl vytvořen testovací nástroj dbgenx, který slouží k naplnění relačních databází testovacími daty. Nástroj dbgenx zvládá vytvářet testovací data na základě uživatelem zadaných a omezení a díky tomu, že pro generování využívá SMT řešitele, je schopen najít řešení i složitě zadaných omezení na generovaná testovací data. Implementovaný nástroj lze využít jak pro vkládání požadovaných testovacích přímo do databáze, tak pro generování výstupu ve formě SQL INSERT INTO příkazů. Částečně je tedy tento nástroj využít pro generování testovacích dat nesouvisejících s databázemi.

Do budoucna by nástroj šlo rozšířit o přesnější generování testovacích dat, například aby generovaná testovací data kopírovala zadanou funkci rozložení pravděpodobnosti, kdy už byl v rámci této práce proveden stručný návrh v kapitole 4.2.6. Další z možných rozšíření, které se už také dostalo do fáze návrhu by bylo rozšíření o generování testovacích dat za účelem pokrytí vstupních domén 4.2.7. Také je určitě v plánu nástroj rozšířit o grafické uživatelské rozhraní a lepší integraci nástroje do platformy Testos.

Literatura

[1] pySMT: A library for SMT formulae manipulation and solving. [Online; navštíveno 14.01.2018].
URL https://github.com/pysmt/pysmt

[2] Skupina Testos: Domovská stránka projektu Testos. FIT VUT v Brně. 2017. [Online; navštíveno 14.05.2018].
URL http://testos.org

[3] Amman, P.; Offutt, J.: *Introduction to Software Testing*. Cambridge University Press, 2008, ISBN 978-0-511-39330-3.

[4] Barrett, C.; Fontaine, P.; Tinelli, C.: The SMT-LIB Standard Version 2.6. 2017.
URL
http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf

[5] Brada, T.: Repozitář projektu sběrnice Testos. 2018. [Online; navštíveno 14.05.2018].
URL https://pajda.fit.vutbr.cz/testos/testos-bus

[6] Chays, D.; Dan, S.; Frankl, P. G.; aj.: A framework for testing database applications. *ACM SIGSOFT Software Engineering Notes*, ročník 25, č. 5, 2000: s. 147–157.

[7] De Moura, L.; Bjørner, N.: Z3: An efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems*, 2008: s. 337–340.

[8] Emmi, M.; Majumdar, R.; Sen, K.: *Dynamic Test Input Generation for Database Applications*. 2007, ISBN 9781595937346, s. 151–162, doi:10.1145/1273463.1273484.

[9] Křena, B.; Vojnar, T.: Automated formal analysis and verification: an overview. *International Journal of General Systems*, ročník 42, č. 4, 2013: s. 335–365.

[10] Kropáč, F.: *Nástroj pro analýzu obsahu databáze pro účely testování softwaru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19446

[11] North, D.: Introducing behaviour driven development. *Better Software Magazine*, 2006.

[12] Ochodek, M.: *Nástroj pro analýzu obsahu databáze pro účely testování softwaru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19259

[13] Solis, C.; Wang, X.: A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, IEEE, 2011, s. 383–387.

[14] The SMT-LIB Initiative: SMT-LIB The Satisfiability Modulo Theories Library. [Online; navštíveno 14.01.2018].
URL http://smtlib.cs.uiowa.edu/solvers.shtml

[15] Znojil, O.: Repozitář projektu db-struct-detectors. 2018. [Online; navštíveno 14.05.2018].
URL https://pajda.fit.vutbr.cz/testos/db-struct-detectors

[16] Červinka, R.: Repozitář projektu combine. 2018. [Online; navštíveno 14.05.2018].
URL https://pajda.fit.vutbr.cz/testos/combine

[17] Želiar, D.: *Nástroj pro generování obsahu databáze pro účely testování softwaru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18065