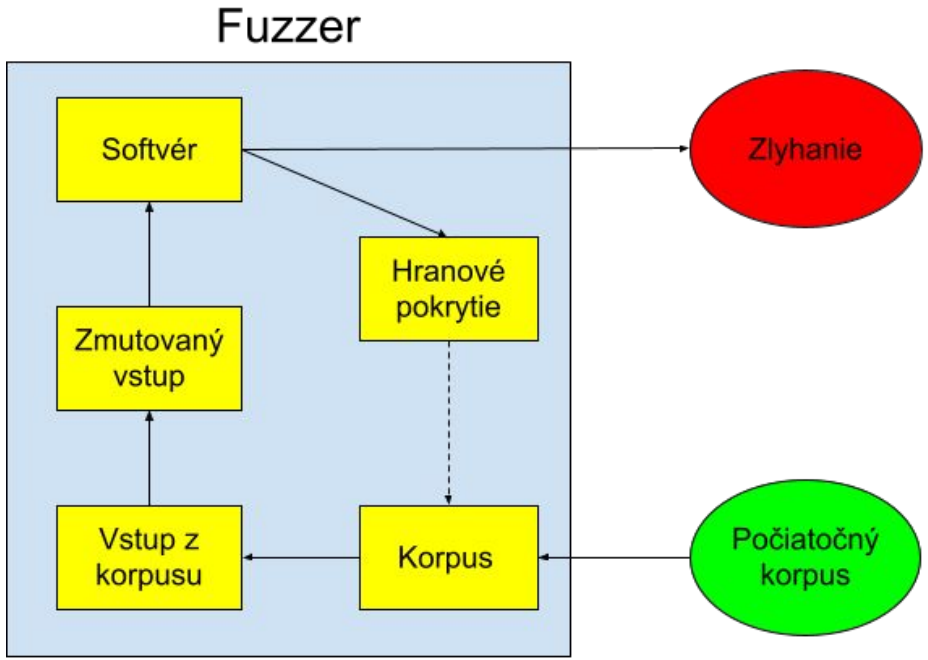
Vylepšenie typového systému medzi-jazyka FuzzIL vo fuzzeri Fuzzilli

RNDr. Richard Ostertág, PhD.

Samuel Sládek

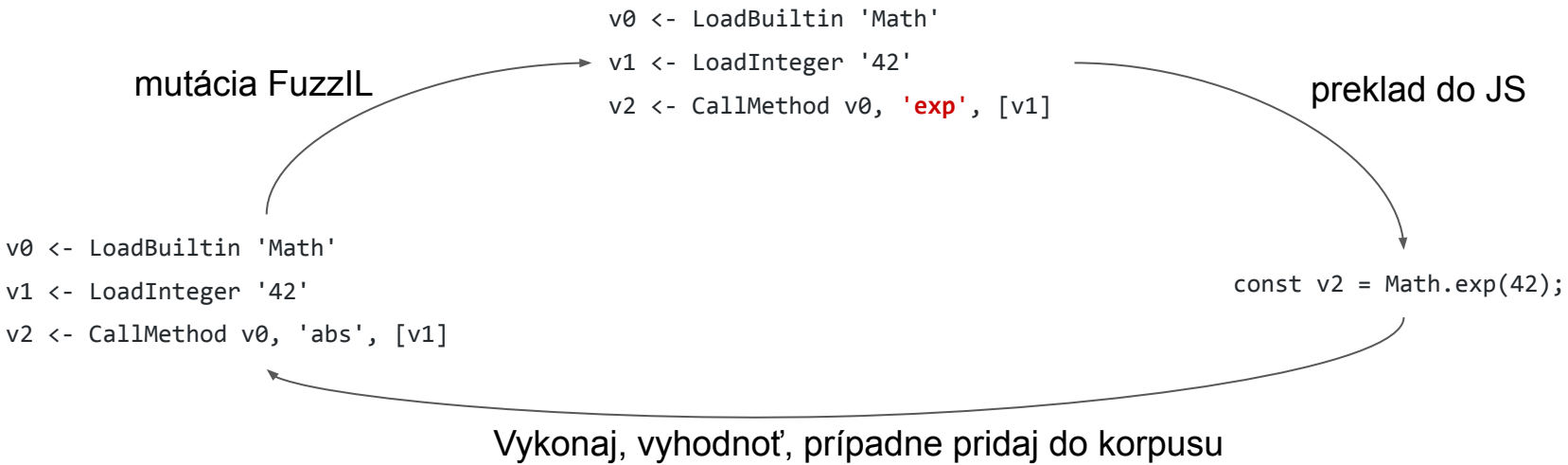
Fuzz testovanie

- hľadanie implementačných chýb a bezpečnostných zraniteľností
- idea simulovaného žíhania
- ľahká škálovateľnosť



Fuzzilli

- Voľne dostupný JavaScript engine fuzzer
- Pokrytím riadený, podporuje generatívny a mutujúci mód
- Používa medzi-jazyk FuzzIL



Odvodzovanie typov

Statické

- Abstraktný interpreter
- Rýchle odvodzovanie
- Približná sémantika JS
- Nepresné typy

Dynamické

- Zbieranie typov počas behu programu
- Pomalé zbieranie
- Skutočné vykonanie skriptu
- Presné typy

Žiadne informácie o typoch počas behu programu

```
const v12 = [new Uint8Array(100), “asdf”];

// v12 = .object(ofGroup: “Array”, ...)

const v14 = v12[0];

// v14 = .unknown (Abstraktný interpreter nepozná typ v14)

v14.???();

// nedokáže vygenerovať užitočnú metódu na v14 :(
```

S informáciami o typoch počas behu programu

```
const v12 = [new Uint8Array(100), “asdf”];

// v12 = .object(ofGroup: “Array”, ...)

const v14 = v12[0];

// v14 = .object(ofGroup: “Uint8Array”, ...)

v14.fill(42);   // :)
```

Ako zbieranie funguje?

```
// Preložené klasicky

const v12 = 42;

const v14 = “foobar”;

const v15 = v12 + v14;
```

```
// Preložené so zbieraním typov

initTypeCollection();

const v12 = 42;

updateType(1, 12, v12);

const v14 = “foobar”;

updateType(2, 14, v14);

const v15 = v12 + v14;

updateType(3, 15, v15);

sendTypesToFuzzilli();
```

Ako zbieranie funguje?

```
for(let v0 = "";v0 < 5;v0++) {
// v0 = .string | .integer
}
const v1 = {}
// v1 = .object(ofGroup: “Object”, ...)
v1.__proto__ = Array.prototype
// v1 = .object(ofGroup: “Array”, ...) + .iterable
```

Adopcia typov pri mutáciách

- Nemôžeme spúšťať zbieranie typov na každom vygenerovanom programe
- Väčšina programu je aj tak nezmenená
- Udržujme čo najviac spoľahlivých typov pri mutovaní programu

```
const v0 = 42;
// v0 = .integer
const v1 = v0 * null;
// v1 = .integer
```

```
const v0 = 42;
// v0 = .integer
const v1 = v0 / null;
// v1 = .unknown
```

Kooperatívny mód

- Kombinácia výhod oboch statického & dynamického odvodzovania typov
- Vkladanie typov z behu programu do Abstraktného interpretera
- Po jednoduchých inštrukciách necháme odvodzovanie typov na Abstraktný interpreter

```
const v0 = {}

updateType(0, 0, v0)

v0.a = 4

updateType(1, 0, v0) // nie je potrebné
```

Limitácie

- Zbieranie dlhých polí sa preskočí

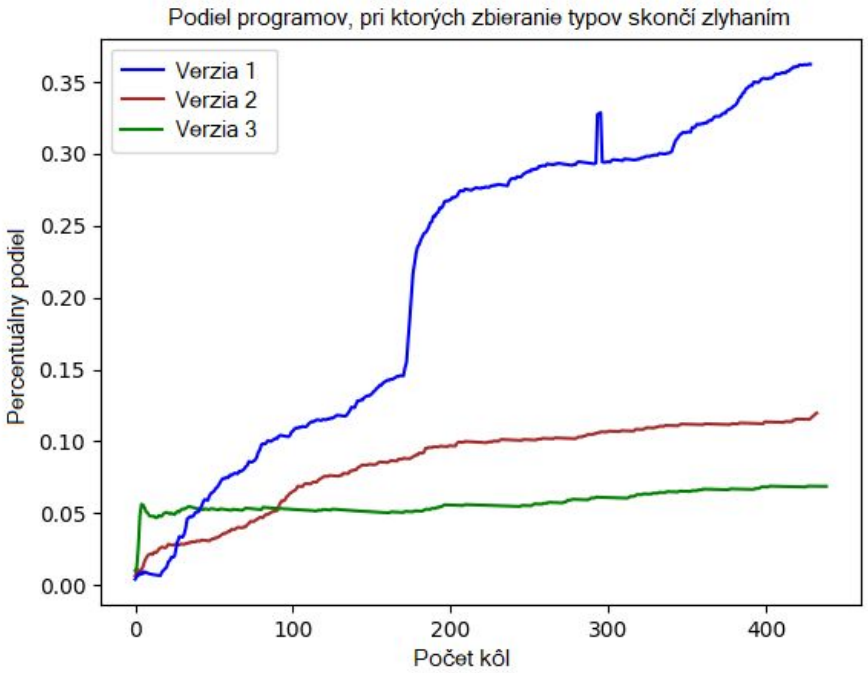
```
const v0 = new UInt8Array(1000000) // rýchla operácia

updateType(0, 0, v0) // pomalá operácia
```

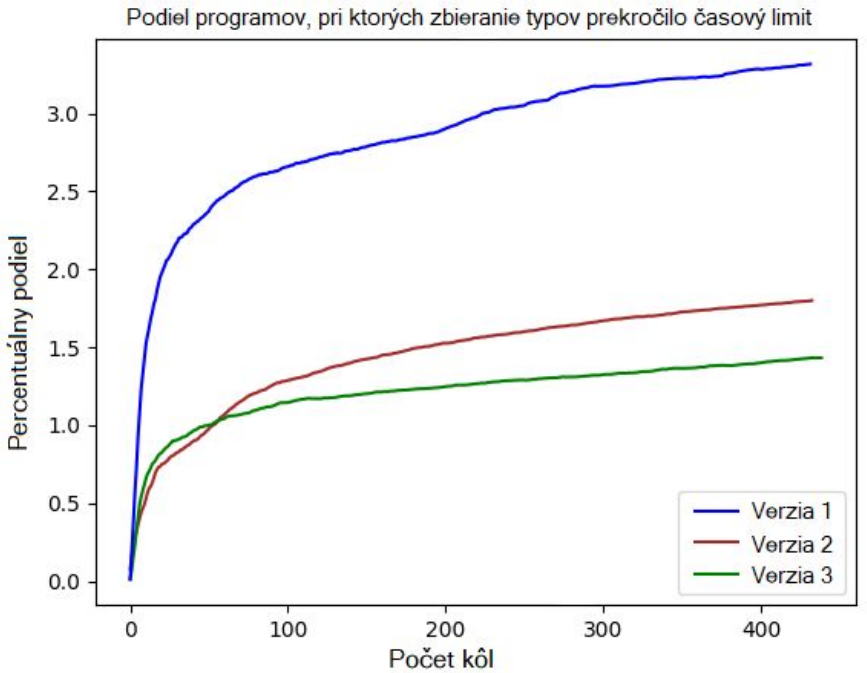
- Vygenerovaný skript vie zmeniť samotný prototyp poľa
  - poškodí nám samotnú štruktúru na uchovanie typov
  - vieme si podstatné veci zálohovať

```
JSON.stringify = 4;
```

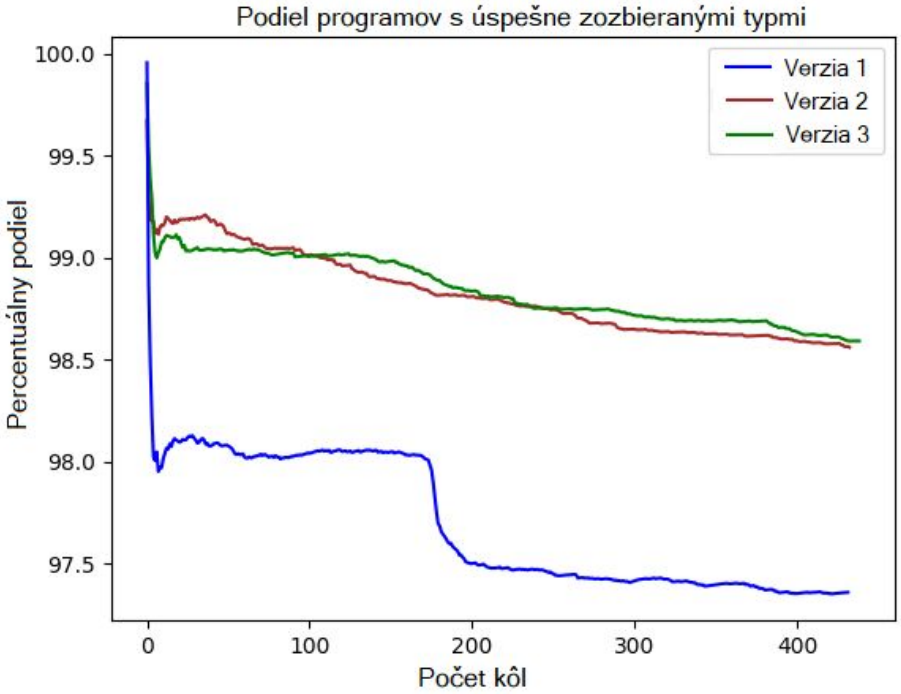
Výsledky

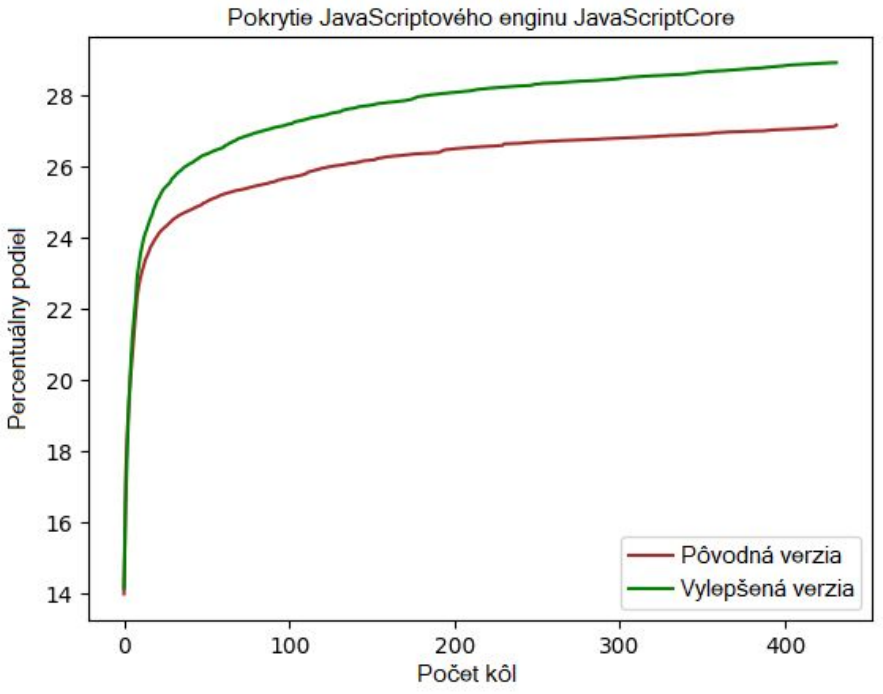
- Nie každé zbieranie typov je úspešné





Výsledky


Podiel programov s úspešne zozbieranými typmi
Verzia 1
Verzia 2
Verzia 3
Percentuálny podiel
100.0
99.5
99.0
98.5
98.0
97.5
0
100
200
300
400
Počet kôl


Pokrytie JavaScriptového enginu JavaScriptCore
Percentuálny podiel
28
26
24
22
20
18
16
14
0
100
200
300
400
Počet kôl
Pôvodná verzia
Vylepšená verzia

Optimalizácie

- Dátová štruktúra na ukladanie typov v programe
  - Nie je potrebné vykonávať program viackrát
  - Efektívne dotazy na typ premennej pri určitej inštrukcii
  - Efektívne dotazy na zmenu typu premennej pri určitej inštrukcii
  - `program.type(of: variable, after: instructionIndex)`
  - Zrýchlenie referenčných skúšok z 30,67s na **16,82s**
- Zdieľanie typových tried na šetrenie pamäte: **200x** menšia spotreba
- Reorganizácia Abstraktného interpretra
  - Efektívnejšie využitie pamäte
  - Zrýchlenie referenčných skúšok z 17,2s na **16,44s**

Optimalizácie

- Implementácia *.minify* módu pre JS prekladač
  - Šetrenie zdieľanej pamäte
  - veľkosť vykonávaných skriptov sa zmenšila o **10%**
- Implementácia typu *.iterable*
  - Vyhýbanie sa generovanie zbytočne nevalidných skriptov

```
const v0 = {}

const v1 = [...v0]
```

Poznámky k realizácii práce

- Implementačná časť práce využívaná spoločnosťou *Google*
- Kód prešiel kontrolou od autora Fuzzilli (vďaka *Samuel Groß*)
- Github
  - Kód pridaný do hlavného repozitára a je plnohodnotnou súčasťou Fuzzilli
  - Diskusie a pripomienky

Priestor na otázky