CONSTRUCTION OF A RANDOM SENTENCE GENERATOR USABLE FOR COMPILER TESTING

Petr Muller
Bachelor Degree Programme (3), FIT BUT
E-mail: xmulle13@stud.fit.vutbr.cz

Supervised by: Tomáš Vojnar
E-mail: vojnar@fit.vutbr.cz

ABSTRACT

This paper discusses construction of a directed random sentence generator, creating grammar sentences of good quality as an input for automated testing of compilers or interpreters. Randomly generated sentences have to be limited by constraints to be useful, yet they still have to keep good randomness characteristics. This paper presents an approach for constructing a generator system which satisfies both criteria.

1 ÚVOD

Jednou z metod automatizovaného testování programů je generování velkého množství náhodně vytvořených dat, jejich použití jako vstupu testovaného programu a analýza chování programu pro tyto vstupy. Tento přístup se nazývá *fuzz testing*. Testuje okrajové případy užití programu a jeho robustnost. Je vhodný pro vyhledávání chyb, nikoliv pro ověřování kvality. Význam má zejména v úvodních iteracích vývoje programu, kdy se vyskytuje nejvíce chyb. V pozdějších iteracích se užitečnost snižuje, protože nelze stanovit jiná akceptační kritéria, než „nenajde chybu“, což nevypovídá o kvalitě programu. Výhodou je fakt, že vystavuje program testovacím případům, které tester–člověk pravděpodobně nepoužije. V toto článku rozebereme návrh generátoru vět, který bude produkovat náhodné věty použitelné pro testování překladačů. Základním kritériem užitečnosti je determinismus chování. Naším cílem je tvorba takových programů, které budou mít při každém spuštění identický výstup, nezávisle na použitém překladači.

2 NĚKTERÉ PROBLÉMY TESTOVÁNÍ PŘEKLADAČŮ

Fuzz testování překladače zcela náhodnými daty není efektivní, testuje jen část kódu překladače, která odmítá neplatný vstup. Efektivní užití metody je obtížnější. Důvodem je netriviální způsob generování vstupů, které podrobí testovaný kompilátor co nejširší podmnožině možných vstupů - programů. Tento prostor je obvykle příliš velký pro kompletní pokrytí. Generátor proto musí umožnit řízení generování vět tak, abychom je mohli přizpůsobit zaměření testování.

2.1 JEDNODUCHÝ GENERÁTOR NÁHODNÝCH VĚT

Jednoduchý generátor vět jazyka definovaného bezkontextovou gramatikou lze získat úpravou konstrukce LL syntaktického analyzátoru [3] technikou rekurzivního sestupu. Jedinou modifikací je záměna rozeznávání symbolů v textu za jejich náhodné generování [2]. Generování postupuje od nejvyššího neterminálního symbolu k terminálům, přičemž rozvoj neterminálu se z možností daných gramatickým pravidlem volí náhodně. Pro generování vět užitečných k testování překladačů je třeba generátor dále zdokonalit.

2.2 PROBLÉMY JEDNODUCHÉHO GENERÁTORU

Praktické užití jednoduchého generátoru ukazuje tři základní problémy. Prvním je fakt, že pro mnoho jazyků neexistuje kompilátor, který by akceptoval věty tvořené jednoduchým generátorem. Některé jazyky, například jazyk C [1], definují podobu programu kromě syntaktických pravidel také sadou omezení (*constraints*). Překladač takového jazyka nemusí přijmout program, i když byl vytvořen podle pravidel gramatiky. Překladače také obsahují implementační limity, které nejsou v gramatice popsány, ale v praxi jsou důvodem pro odmítnutí programu.

Druhý problém představuje hluboká hierarchická struktura některých neterminálů (typicky to jsou výrazy [1]). Pokud je v hierarchii větší množství neterminálů, které mají několik přímo rekurzivních pravidel rozvoje, pak je výsledný strom velmi komplikovaný, protože každý takový neterminál se pravděpodobně několikrát rozvine rekurzivně, než sestoupí v hierarchii níže. Protože se jedná o několikanásobné uplatňování každého pravidla, nepřináší tato složitost v testování užitek. Je velmi pravděpodobné, že takto implementovaný generátor bude náchylný na problémy spojené s extrémní rekurzivitou algoritmu generování (např. vyčerpání zdrojů).

Třetím problémem je výskyt nežádoucích elementů v generovaném programu. Příkladem jsou nekonečné smyčky, neinicializované proměnné, nebo nevolané funkce. Tyto komplikace snižují užitečnost generovaného případu a je nutné omezit generování tak, aby k nim nedocházelo.

3 NÁVRH KONSTRUKCE ŘÍZENÉHO GENERÁTORU

Náš návrh dělí řízený generátor na dva moduly. První je generátor kódu, jehož vstupem je gramatika jazyka, popsaná v Backus-Naurově formě rozšířené o prvky nutné k popsání jazyka generovaných vět, např. volitelné symboly. Druhý modul je tzv. *director*, který uchovává informace o sémantice programu a uplatňuje konfigurovatelná omezení generovaných symbolů.

3.1 KONSTRUKCE GENERÁTORU VĚT

Konstrukce námi vytvořeného generátoru je triviální. Generátor generuje symboly shora dolů. Nejprve oznámí directoru symbol, který začíná generovat. Director zašle zpět seznam možných rozvojů s pravděpodobnostmi, s jakými se generují. Podle pravděpodobnosti se zvolí jeden rozvoj a rekurzivně se vygeneruje. Na konci zašle generátor directoru finální podobu symbolu.

3.2 KONSTRUKCE DIRECTORU

Modul director slouží k eliminaci problémů popsaných v podsekci 2.2. Umožňuje generovat jen užitečnou podmnožinu vět testovaného jazyka. Odděluje pravidla pro získání této podmnožiny od gramatiky, přestože by do ní mohla být teoreticky zapracována přímo. Pak by ale bylo

třeba měnit gramatiku při každé změně parametrů, což je neefektivní a je zde možnost zanesení chyby. Director pracuje následovně: přijme od generátoru seznam rozvojů symbolu daný gramatikou. Na základě konfigurace přiřadí jednotlivým rozvojům pravděpodobnosti. Jednotlivé rozvoje z gramatiky může vypustit nebo modifikovat dle konfigurace, např. zajistit vystoupení z cyklu v případě jeho neukončení po určitém počtu průchodů. Název generovaného symbolu zařadí do zásobníku, který udržuje podobu větve derivačního stromu a umožnuje na základě informací o předcích rozhodovat o dalším průběhu generování. Poté, co seznam projde úpravami podle všech pravidel, je odeslán generátoru. Generátor oznamuje každý hotový symbol, který director sémanticky zpracuje na základě pravidel. Sémantickým zpracováním rozumíme zařazení do vnitřních struktur directoru, na jejichž základě se mohou modifikovat další generované věty. Můžeme tak zajistit, aby se např. vygenerovaná proměnná inicializovala v prologu generovaného programu, vypsala v epilogu, a použila se v později vygenerovaném příkazu.

Director implementuje následující konfigurovatelné vlastnosti:

Vyloučení symbolů. Pokud symbol obsahuje rozvoj obsahující vyloučený symbol, pak se takovému rozvoji přiřadí pravděpodobnost 0 v případě povinného výskytu vyloučeného symbolu nebo se symbol jen vypustí v případě, že se vyskytuje jako nepovinný.
Přiřazení specifické pravděpodobnosti rozvojům určitého symbolu.
Pooling rozvojů. V rámci určitého symbolu je definován rozsah, který se užitím různě složitých rozvojů různě rychle vyčerpává. Je tak určena maximální „délka“ symbolu.
Pravděpodobnost rozvojů. Pravděpodobnosti se mění dle předchozích rozvojů podle pravidel.
Změny rozvojů. Možnost předefinovat podobu rozvoje neterminálu gramatiky, např. pro výpisy nebo přidání proměnných zajišťujících ukončení cyklu nebo rekurze.
Omezení. Možnost omezit přítomnost symbolu jen na určená místa v generovaných stromech.
Prolog a epilog. Modulární generování prologu/epilogu podle vygenerovaných symbolů.

Tyto vlastnosti zajišťují možnost konfigurace, umožňující omezení dopadů problémů jednoduchého generátoru na minimum. Zároveň poskytují tvorbu dostatečně náhodných vět, aniž by došlo k omezení užitečnosti generovaných programů pro testování. Jde o robustní řešení, umožňující i zaměření generovaných programů pro testování jednotlivých částí překladače.

4 ZÁVĚR

Generátor je navržen s ohledem na robustost a užitečnost vět pro testování překladačů. Užívá komplikovaný systém konfigurací. Tato složitost může být vyvážena automatickým generováním některých pravidel. Výsledky prototypové implementace řízeného generátoru se zdají být poměrně slibné. Vygenerované věty se dají přeložit a po implemetaci sémantického analyzátoru snad bude dosaženo i vyhnutí se nejednoznačným příkazům a sémantickým chybám.

REFERENCE

[1] ISO/IEC 9899 TC3: Programming Languages - C, 1999, s. 5

[2] Schwartz, R.: Writing Nonsense With Perl. Linux Magazine, 9/1999. Dokument dostupný na URL http://www.linux-mag.com/id/309 (březen 2008)

[3] Aho, A. V., Lam, M. S., Sethi, R, a Ullman, J. D. Compilers: Principles, Techniques, and Tools (2nd edition). Addison-Wesley, Boston, MA, USA, 2006. ISBN-0321486811