AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Studio projektowe 2

Benchmark solvera InKreSAT

29 marca 2021

Autorzy:

Mateusz Grzeliński

Przemysław Michałek

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Spis treści

1 Wprowadzenie

Celem projektu jest zbadanie wydajności automatyczych metod dowodzenia twierdzeń InKreSAT w kontekście formuł żywotnościowych i bezpieczeństwa, przedstawionych jako problem SAT (boolean satisfiability problem), a dokładniej TL (Temporal Logic). Na początku zostaje wygenerowna formuła SAT, która zostaje rozwiązana przez badany prover. Badany jest czas wykonania, rezultat (czy SAT jest spełnialny), użycie pamięci RAM. Generowana formuła SAT jest modyfikowana ze względu na między innymi długość formuły, ilość zmiennych.

Prover traktowany jest jako czarna skrzynka (blackbox), jego parametry są modyfikowane z poziomu linii komend.

1.1 Żywotność i bezpieczeństwo formuł

Żywotność systemu to cecha, która zapewnia, że coś dobrego na pewno w końcu się wydarzy. Formuła żywotnościowa gwarantuje, że istnieje co najmniej jedno wydarzenie, dla którego formuła będzie spełniona.

Bezpieczeństwo systemu to cecha, która zapewnia, że nic złego nigdy się nie stanie. Formuła bezpieczeństwa gwarantuje, że dla każdego wydarzenia, formuła bezpieczeństwa nigdy nie zostanie pogwałcona.

Żywotność oraz bezpieczeństwo mogą zostać wyrażone na przykład w postaci formuły logicznej logiki temporalnej, czyli problemu SAT. Logika temporalna umożliwiająca rozważanie zależności czasowych bez wprowadzania czasu explicite. Operatory czasu logiki temporalnej to:

- `[]p` - p jest spełnione we wszystkich stanach (zawsze)
- `<>p` - p jest spełnione przynajmniej w jednym stanie (kiedyś)
- `()p` - p jest spełnione w stanie następnym po stanie odniesienia (następny)
- `pUq` - p jest spełnione dopóki nie jest spełnione q (dopóki)

1.1.1 Przykład: zdawanie egzaminu

Problem: zalicz egzamin, aby zdać kurs

Warunek bezpieczeństwa: jeżeli podejmujesz się egzaminu, zalicz go

Warunek żywotnościowy: kiedyś musisz podejść do egzaminu

1.1.2 Przykład: światła na skrzyżowaniu

Problem: samochody chcą przejechać przez skrzyżowanie

Warunek bezpieczeństwa: tylko jedno światło powinno być zielone

Warunek żywotnościowy: każde światło powinno kiedyć zmienić się na zielone

2 Benchmark

Problem benchmarku w strategii blackbox sprowadza sie do wykonania podprogramu z odpowiednimi opcjami z poziomu linii komend. Wejście oraz testy benchmarka ustawiane są poprzez plik konfiguracyjny, sekcja 2.3. Wejściem testu jest zbiór formuł SAT zapisanych na dysku w formacie TPTP w postaci pliku tesktowego. Test polega na podaniu pliku TPTP do provera. W razie potrzeby nastąpi automatyczna konwersja do odpowiedniej składni za pomocą dostępnych translatorów. Dla każdego provera dostępne są statystyki:

- czas wykonania
- użycie pamięci RAM
- spełnialność formuły

Więcej statystyk może zostać zebrany przez parser, który bada wyjście provera (sekcja 2.2).
Zbadane zostaną statystyki proverów ze względu na następujące właściwości formuły SAT:

- SAT type (CNF, FOF)
- number of clauses (CNF) - w składni TPTP jest to liczba użytych słów `cnf`
- maximal clause size
- number of variables

2.1 InKreSAT

InKreSAT to zautomatyzowane narzędzie udowadniające dla logiki temporalnej stworzone przez Marka Kaminskiego oraz Tobiasa Tebbi. Redukuje podany modalny problem SAT do postaci Booleanowego problemu SAT, który następnie rozwiązuje solverem SAT. InKreSAT optymalizuje wcześniejsze metody wykonując kolejne kroki inkrementacyjnie, czyli co każdy krok pobiera dane z używanego solvera SAT i używa ich do optymalizacji następnego kroku.

InKreSAT dostępny jest jako plik wykonywalny, jako argumenty przyjmuje pliki o rozszerzeniu fml. Ilość opcji dostępnych z lini komend jest minimanlna, jedyną ważną opcją z punktu widzenia benchmarka jest `-v n  set verbosity level to n (min: 0, default: 1, max: 3)`

Oficjalna strona internetowa https://www.ps.uni-saarland.de/~kaminski/inkresat/

```
begin
[](~[]<>p | ~[](~p | []q))
end
```

Listing 1: Przykład pliku wejściowego .fml

```
variables added: 1
clauses added: 1
SAT cycles: 1
pattern store hits: 0
pattern store misses: 0
time: 6.29425048828e-05
SATISFIABLE
```

Listing 2: Przykład wyjścia InKreSAT

2.2 Parser

Zadaniem parsera jest wydobycie dodatkowych informacji statystycznych o przebiegu działana proverów, na podstawie ich wyjścia. Statystyki zostaną podane w formacie json.

2.3 Użycie i konfiguracja

Najpierw definiowana jest lista wejść (`testInput`). Wejście to zbiór plików, które można jednoznacznie zidentyfikować za pomocą nazwy (`name`). Następnie definiowana jest lista zestawów testowych (`testSuite`). Zestaw testowy definiuje parametry wspólne dla kilku przypadków testowych (`testCase`) np. ścieżka do pliku wykonywalnego. Każdy zestaw testowy posiada listę przypadków testowych. Każdy przypadek testowy definiuje w jakim formacie oczekuje wejście. Jeśli formaty są różne, konflikt jest rozwiązywany przy pomocy dostępnych translatorów. Opcje do testowania są definiowane jako lista. Plik wejściowy może zosać podany przez standardowe wejście, przez opcję lini komend lub jako ostatni argument w komendzie.

```
testSuite.executable testSuite.options testSuite.testCase.options
↪  [input_after_option file_path] [file_path]
```

2.3.1 Wspierane funkconalności

- ścieżka do pliku wykonywalnego może być podana w pliku konfiguracyjnym, lub może być zawarta w zmiennyj środowiskowej `PATH` (ścieżka w pliku ma pierwszeństwo)
- definiowanie opcji linii komend do testowania pliku wykonywalnego
- definiowanie listy źródeł do testów. Źródłem do testów mogą być tylko pliki tesktowe
- definiowanie które wejścia mają byś przetestowane w przypadku testowym
  - testuj tylko wymienione - opcja `include_only` ,
  - testuj wszystkich oprócz - opcja `exclude` ,
  - testuj wszystkie zdefiniowane wejścia - nie podając żadnej z opcji
- pozycja nazwy pliku źródłowego może być ustawiona w następujący sposób
  - domyślnie plik podawany jest na standardowe wejście

    - podaj plik jako ostatni argument `input_as_last_argument`
    - podaj plik jako argument po opcji `input_after_option`
- TODO: wyniki zapisywane są jako plik *json* do katalogu wyściowego zdefiniowanego w pliku konfiguracyjnym.

2.4 Diagramy

3 Generator formuł logicznych

Losowy generator formuł SAT. Generator może generować formuły CNF (Conjunctive Normal Form) oraz . Generator może generować formuły w formacie DIMACS oraz TPTP.

4 Zestaw testowy

Problem: Jak ilość zmiennych wpływa na czas wykonania?

Zestaw to formuły CNF różnej długości

- liczba klauzul to 100, 200, 500, 1000, 2500, 5000
- stosunek liczby zmiennych do klauzul wynosi 2, 3, 4, 5, 10 (2 oznacza, że jeśli liczba klauzul to 100, wtedy liczba zmiennych to 200)

Problem: Jak stosunek atomów bezpieczeństwa do atomów żywotnościowych wpływa na czas wykonania?

Zestaw to formuły CNF różnej długości

- liczba klauzul to 100, 200, 500, 1000, 2500, 5000
- stosunek atomów bezpieczeństwa do atomów żywotnościowych wynosi 2, 3, 4, 5, 10
- maksymalna długość klauzuli to ilość wszystkich zmiennych / ilość klauzul

Problem: Jak udział k-SAT i zmiennych wpływa na czas wykonania?

Zestaw skupia się na k-SAT:

- liczba klauzul to 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000
- klauzule są postaci:
  - 1,5,10,20-SAT – kada z nich stanowi 25% wszystkich klauzul (po równo)
  - 1,5,10,20-SAT – 1-SAT to 1% (minimum 1), 5,10,20-SAT po równo
  - 1,5,10-SAT – po równo
  - 1,5,10,20-SAT – 20-SAT to 1% (minimum 1), 1,5,10-SAT po równo
  - 5,10,20-SAT – po równo

5 Wnioski

Celem projektu było zbadanie które z czynników formuł logicznych wpływają najbardziej na działanie provera InKreSAT. Aby odpowiedzieć na to pytanie napisany został

generator formuł o zadanych parametrach które następnie były przepuszczane przez prover z użyciem strategii blackbox. Na wyjściu otrzymane zostały pliki JSON z parametrami wejścia takimi jak: liczba klauzul, atomów, predykatów, funktorów, zmiennych oraz parametrami wyjścia: czasem wykonania, maksymalnym zużyciem pamięci (peak memory) oraz statusem (spełnialna, niespełnialna, timeout). W celu ograniczenia czasu wykonywania się benchmarków wprowadzono górną granicę czasu na każdy test case: 300 sekund.

5.1 Zestaw 1

Jak widać na wykresach załączonych poniżej według oczekiwań największy wpływ na czas wykonywania/używaną pamięć największy wpływ ma ilość klauzul.

5.2 Zestaw 2

W zestawie 2 badano wpływ stosunku ilości atomów bezpieczeństwa do ilości atomów żywotności. W grupach test case'ów o tych samych ilościach klauzul zadano różne stosunki ilości atomów bezpieczeństwa do ilości atomów żywotności.

5.3 Zestaw 3

W zestawie 3 badano wpływ udziału klauzul k-SAT na czas wykonywania/pamięć w zależności od k. Zadano klauzule o różnym stopniu rozłożenia spośród wartości k ϵ {1,5,10,20}.

Skrót

ATP Automated Theorem Proving. ATP systems are enormously powerful computer programs, capable of solving immensely difficult problems, http://www.tptp.org/OverviewOfATP.html

BNF Backus normal form

CNF conjunctive normal form

DNF disjunctive normal form

FOF First Order Formula

FOL First Order Logic

ILP Integer Linear Programming

IP Integer Programming

LADR library written in c, shipped with Prover9. It also defines input syntax required by Prover9

LTL jedna z logit temporalnych

prover SAT solver

PRP Propositional Logic

SAT boolean satisfiability problem

SMT Satisfiability Modulo Theory

SMTLIB SMT-LIB is an international initiative aimed at facilitating research and development in Satisfiability Modulo Theories (SMT). Homepage http://smtlib.cs.uiowa.edu/index.shtml

Temporal Logic Temporal Logic

TFF Typed First-order Logic

THF Typed Higher-order Logic

TMTP Thousands of Models for Theorem Provers

TPI TPTP Process Instruction - source http://www.tptp.org/Seminars/TPI/Abstract.html

TPTP Thousands of Problems for Theorem Provers, official wesite http://www.tptp.org

TPTP2X The TPTP2X utility is a multi-functional utility for reformatting, transforming, and generating TPTP problem files.

TPTP4X The TPTP4X utility is a multi-functional utility for reformatting, transforming, and generating TPTP problem files. It is the successor to the TPTP2X utility, and offers some of the same functionality, and some extra functionality. TPTP4X is written in C, and is thus faster than TPTP2X.

TSTP Thousands of Solutions from Theorem Provers