Тестирующая система

1 Начало

Может быть многие уже знают, что с начала апреля работает тестирующая онлайн-система. Она умеет проверять готовый архив перед отправкой на ошибки. Всё, что нужно, для того, чтобы начать ей пользоваться, это зайти на страницу 95.85.44.51 (или теперь она же доступна по адресу uchu.retorta.su. И даже по https). Войти на свою страницу, и там с самого верха есть раздел "Посылки" с ссылкой на форму загрузки. После загрузки ваш архив автоматически протестируется и можно будет со страницы посылок посмотреть отчёт по нему. Все замечания системы стоит воспринимать как рекомендации. Искуственного интелекта в ней нет, и ошибаться она тоже может. Но обратить внимание на её замечания стоит.

2 Зачем это нужно

В первую очередь система тестирования писалась для автоматического нахождения банальных ошибок. Около половины (если не больше) ошибок, которые допускаются в генераторах довольно типичны. Более того, на эти ошибки вполне можно проверять в автоматическом режиме. Так зачем на это тратить человеческие ресурсы? Ну а также система работает круглосуточно и 7 дней в неделю. И отвечает значительно быстрее, чем человек (ей нужно 25 секунд на каждый генератор). Ну и в дополнение к этому сейчас появились проверки, которые человек скорее всего не обнаружит. После того, как я перетестил все имеющиеся у меня генераторы в 25 из них были найдены критически проблемы (что приводит к некорректным условиям, аппеляциям и вообще неприятно).

3 Что в ней есть

3.1 Компиляторы

Прежде всего система проверяет генератор на компилируемость. Для этого я стандартно использую gcc (g++-4.9 -fpermissive -std=c++11 -Wextra -Wpedantic -Wall) и вывожу его предупреждения.

Также я справился подружить linux и компилятор из MVS. (cl.exe /EHsc /Wall /W4) и его вывод также появляется в предупреждениях.

3.2 Запуск

Дальше происходит сначала запуск программы, а потом и сборка tex из получившегося файла. Каждое из этих действий запускается с timeout. Так что возможно если тестирование падает на этом месте, то ваша программа делает это очень долго. (Предупреждения из tex также попадают в отчет).

3.3 Анализ по регулярным выражениям

Эта проверка просто ищет места в коде, которые соответствуют некоторым регулярным выражениям. Помогает найти что-то типа _frac_|2|_frac (нужно заменить на _frac2); task|'Д'|"еревня"(Стоит объединить в одну строчку, так как это станет работать быстрее. Операция | не так мало стоит). (И ещё около 20 других проверок)

3.4 PVS-Studio

Также я проверяю код статическим анализатором. Удобных и полезных их не так много. (cppcheck, например, не находит ничего в подавляющем большинстве случаев). Поэтому я использую PVS-Studio. В него иногда отлавливаются ошибки, которые не так просто найти даже при аккуратном вычитывание кода. Подробные разъяснения кодов ошибок также можно найти на сайте. (Ну и вообще довольно классный инструмент, полезно знать, что такое бывает).

Надеюсь, что они не узнают, что я использую их в не самых учебных целях

3.5 Проверка видов и выражений

Также я использую самописную систему проверки выражений. (Парсер использую из clang). Для каждого генератора она строит дерево видов. (Какой участок кода для каких видов выполняется). Она помогает найти разные нетривиальные случае неверных выражений. Например вот такой код: if(vid==1){if(vid==2){}} вызовет предупреждение. Так как условие во втором if всегда ложно. Или реальный пример, который прошёл все проверки: (vid%12>5&&vid%2<8). Заметить его при обычной проверке крайне сложно, а вот тестирующая система умеет такое находить, и говорит, что предыдущее выражение всегда истинно.