Символьное исполнение программ

Предлагается написать движок для символьного исполнения C-кода.

1.1 Трассер

Трассой программы называется журнал событий, произошедших во время выполнения программы. События могут быть следующие:

- Сложение двух чисел, вычитание двух чисел, умножение двух чисел и унарный минус
- По какой ветви условного оператора пошла программы и какое было условие

Каждая запись в трассе имеет следующий вид:

- Инструкции присваивания:
  lvalue = rvalue + rvalue
  lvalue = rvalue - rvalue
  lvalue = rvalue * rvalue
  lvalue = rvalue
  lvalue = - rvalue
  где lvalue и rvalue - это пара (адрес, значение). Адрес - id переменной. Если адрес равен 0 - то это константа. Если величина равна «_», то она не имеет для нас значения и мы можем её игнорировать. Значение в lvalue всегда _ (так как оно всегда будет перезаписанно).
  Инициализация перменной происходит следующим образом:
  (m,_) = (0,value)
  (m,_) = (type)(xn,_)
  где type - тип данных, xn - имя переменной. Значение value - конкретное значение данной переменной.
  Формат input файла такой:
  sat
  (= x1 -10)
  (= x2 -20)
  здесь sat - ключевое слово, (= x1 -10) - присвоить первой переменной -10, (= x2 -20) - присвоить второй переменной -20.
- Операции имеют вид трехадресного кода, т.е. например y = x * 2 + 10 разбиваются на 2 операции:
  temp = x * 2
  y = temp + 10
  Соответственно, в журнале будет 2 записи на одну такую строчку.

- Инструкции ветвления:
  then:branchid true
  then:branchid rvalue < rvalue
  then:branchid rvalue <= rvalue
  then:branchid rvalue > rvalue
  then:branchid rvalue >= rvalue
  then:branchid rvalue == rvalue
  then:branchid rvalue != rvalue

  else:branchid true
  else:branchid rvalue < rvalue
  else:branchid rvalue <= rvalue
  else:branchid rvalue > rvalue
  else:branchid rvalue >= rvalue
  else:branchid rvalue == rvalue
  else:branchid rvalue != rvalue
  then - значит условие было истинным, else - ложным; branchid - номер ветвления.

Для такого кода:

```
1  #include <stdio.h>
2  #include <cute.h>
3
4  int dbl(int x) {
5      return 2 * x;
6  }
7  main() {
8      int x; int y;
9
10     CUTE_integer(x);
11     CUTE_integer(y);
12
13     printf("x=%d y=%d\n",x,y);
14     int z = dbl(x);
15     if(z==y) {
16         if(x != y+10) {
17             printf("I am fine here\n");
18         } else {
19             printf("I should not reach here\n");
20         }
```

```
21          }
22  }
```

на входных данных данных 0 0 будет выведена такая трасса:
(140733652199464,_) = (0,_)
(140733652199500,_) = (140733652199488,0)
(140733652199456,_) = (0,_)
(140733652199448,_) = (0,4202148)
(140733652199500,_) = (int)(x1,_)
(140733652199440,_) = (0,_)
(140733652199496,_) = (140733652199480,0)
(140733652199432,_) = (0,_)
(140733652199424,_) = (0,4202148)
(140733652199496,_) = (int)(x2,_)
(140733652199416,_) = (0,4202152)
(140733652199408,_) = (0,_)
(140733652199404,_) = (140733652199500,0)
(140733652199392,_) = (0,_)
(140733652199388,_) = (140733652199496,0)
(140733652199376,_) = (0,_)
(140733652199372,_) = (140733652199500,0)
(140733652199276,_) = (140733652199372,_)
(140733652199472,_) = (0,2) * (140733652199276,0)
(140733652199476,_) = (140733652199472,0)
(140733652199360,_) = (0,_)
(140733652199356,_) = (140733652199496,0)
then:1 (140733652199476,0) == (140733652199356,0)
(140733652199344,_) = (0,_)
(140733652199340,_) = (140733652199496,0)
(140733652199336,_) = (140733652199340,0) + (0,10)
(140733652199328,_) = (0,_)
(140733652199324,_) = (140733652199500,0)
then:3 (140733652199324,0) != (140733652199336,10)
(140733652199312,_) = (0,4202163)
1.2 Тесты
Тесты - прогрммы, которые вы будите анализирвать. После объявления переменных пишутся одни из этих макросов:
CUTE_integer(x)
CUTE_unsigned_integer(x)
CUTE_character(x)
CUTE_unsigned_character(x)

CUTE_short(x)
CUTE_unsigned_short(x)
CUTE_long(x)
CUTE_unsigned_long(x)
CUTE_long_long(x)
CUTE_unsigned_long_long(x)
Когда код из пункта (1.1) будет запущен на входных данных
sat
(= x1 -10)
(= x2 -20)
то переменной x будет присвоено -10, переменной y будет присвоено -20
1.3 Движок
Конечная цель свзки трассер+движок - покрыть все пути выполнения программы. Путь выполнения программы - комбинация выбора ветвей условных операторов. Вам будет предоставлен трассер, вам нужно написть движок.
По трассе ваш движок должен сгенерировать новый input, такой что будет покрыт новый путь выполнения программы. Если движок не может сгенерироват новый input, то он должен закончить свою работу с кодом возврата, не равным нулю. Не возбраняется создавать промежуточные файлы для коммуникации двух разных запусков вашего движка.
Для решения задачи предлагается реализовать символический интерпретатор с символическими переменными. Движок должен по трассе собрать ограничения на символьичесие переменные после чего с помощью солвера сгенерировать новые ограничения. Для разрешения ограничений можно использовать любой солвер. Движок может быть написан на любом языке программирования.
1.4 Запуск
Для запуска нужно использовать docker (везде потребуется sudo): docker build . -t ctask
docker run -it –entrypoint bash ctask
но я предпочитаю подключатся к контейнеру через ssh:
docker run -d ctask
docker inspect «id контейнера» | grep ip -i это для поиска ip контейнера
ssh ctask@ip_контейнера
пароль - ctask
Внутри в папке /home/ctask/assignment/cute/tests будут лежать тесты, на которых будет тестироваться ваш движок. Инструментация кода (например, testme.c) запускается так: ../cutec testme.c
После чего вы получите файл testme.exe - файл, который считывает значения переменных из input, выполняет код и печатает трассу в файл

trace.
Запуск трассера в связке с вашим движком (назовём его driver.out):
../cute ../mycute testme.exe -i 3
где -i 3 - запуск на не более, чем 3 трассах. В конечном итоге вам нужно запустить testsall.in:
./testall driver.out