Gem #121: Команды для точек прерывания - Часть 1

Автор: Jerome Guitton, AdaCore

Краткое содержание: В данном Gem #121 предоставляется простая демонстрация применения команд для точек прерывания в GDB, используемых с целью проверки некоторых параметров в ходе выполнения отлаженной программы.

Давайте начнём…

Рассмотрите эту базовую модель Банка:

```
package Bank is
   procedure Open_Account (Cash : Integer);
   procedure Deposit (Cash : Integer);
   procedure Withdraw (Cash : Integer);
end Bank;
```

Этот банк имеет только один счет, который должен быть открыт до совершения какой-либо операции. Единственным клиентом этого банка будет основная подпрограмма `P`, которая открывает счет и осуществляет несколько операций:

```
with Bank;
procedure P is
begin
   Bank.Open_Account (100);
   Bank.Deposit (45);
   Bank.Withdraw (55);
   Bank.Deposit (80);
end P;
```

Реализация пакета банка проста, он также содержит дополнительную транзакцию во время разработки, что соответствует специальному предложению банка (и уже можно увидеть, что это вызовет проблему):

```
package body Bank is
   Balance : Integer;
   procedure Open_Account (Cash : Integer) is
   begin
      Balance := Cash;
   end Open_Account;
   procedure Deposit (Cash : Integer) is
   begin
      Balance := Balance + Cash;
   end Deposit;
   procedure Withdraw (Cash : Integer) is
   begin
      Balance := Balance - Cash;
   end Withdraw;
  -- To celebrate the inauguration of the bank at elaboration time, your
  -- account is credited with $100.
   Special_Offer : constant Integer := 100;
begin
   Deposit (Special_Offer);
end Bank;
```

Существует неявное свойство, что счет должен быть открыт только один раз, и до того, как произойдет какая-либо транзакция; будет ли это специальное предложение нарушать это свойство? Это хорошая возможность увидеть, как GDB может помочь нам в проверке свойств этой формы и выполнить перерыв в выполнении, когда свойство нарушается во время выполнения. Для этого можно использовать язык сценариев GDB вместе с командами точек останова.

Первое что необходимо сделать это – установить точки останова для банковских операций:

```
(gdb) break open_account
Breakpoint 1 at 0x401592: file bank.adb, line 7.
(gdb) break deposit
Breakpoint 2 at 0x4015a2: file bank.adb, line 12.
(gdb) break withdraw
Breakpoint 3 at 0x4015b8: file bank.adb, line 17.
```

Однако мы не хотели бы останавливаться ни на одной транзакции, а реагировать только на недействительной. На каждой точке останова мы хотели бы проверить, удерживаются ли наши инварианты, продолжить, если они это сделают, и остановить с ошибкой, если они этого не делают.

Во-первых, свойства, которые мы хотим проверить, требуют поддержания состояния: аккаунт открыт. Это может быть представлено вспомогательной переменной GDB, которая может быть создана следующим образом:

```
set variable $account_open := False
```

Затем мы можем прикрепить небольшую программу, которая будет выполняться при каждом достижении точки останова. например для точки останова 1 на `Open_Account` :

```
(gdb) commands 1
>if $account_open = True
 >echo error: account opened twice
 >else
 >set variable $account_open := True
 >printf "(info) account created with $%d
", cash
 >continue
 >end
>end
```

Эта команда возобновит работу программы, если учетная запись еще не была открыта, и остановится с сообщением об ошибке, если мы попытаемся открыть ее во второй раз. Для `Withdraw` и `Deposit` можно использовать одну и ту же программу:

```
(gdb) commands 2
>if $account_open = False
 >echo error: someone tries to deposit before opening an account
 >else
 >printf "(info) deposit: $%d", cash
 >continue
 >end
>end
(gdb) commands 3
>if $account_open = False
 >echo error: someone tries to withdraw before opening an account
 >else
```

```
 >printf "(info) withdrawal: $%d", cash
 >continue
 >end
>end
```

Теперь давайте запустим программу и поищем инвариантные нарушения. Одно происходит сразу:

```
(gdb) run
```

Breakpoint 2, bank.deposit (cash=100) at bank.adb:12 12 Balance := Balance + Cash; error: someone tries to deposit before opening an account (gdb)

ошибка: кто-то пытается внести депозит перед открытием счета (gdb)

Действительно, специальное предложение зачисляется до открытия счета:

```
(gdb) up
#1  0x004015da in  () at bank.adb:25
25         Deposit (Special_Offer);
```

Давайте продолжим искать другие нарушения. Программа выходит нормально, что означает, что другие нарушения не были обнаружены.

```
(gdb) c
Continuing.
```

Breakpoint 1, bank.open_account (cash=100) at bank.adb:7 7 Balance := Cash; (info) account created with $100

Breakpoint 2, bank.deposit (cash=45) at bank.adb:12 12 Balance := Balance + Cash; (info) deposit: $45

Breakpoint 3, bank.withdraw (cash=55) at bank.adb:17 17 Balance := Balance - Cash; (info) withdrawal: $55

Breakpoint 2, bank.deposit (cash=80) at bank.adb:12 12 Balance := Balance + Cash; (info) deposit: $80 [Inferior 1 (process 7472) exited normally]

Язык сценариев GDB предоставляет различные команды управления потоком; Вы можете найти их полную документацию в разделе "Command files" («Командные файлы») в руководстве пользователя GDB. Мы видели, что этот язык GDB, наряду с вспомогательными переменными и командами точек останова, предоставляет простой способ проверки некоторых простых динамических свойств в ходе выполнения. Чтобы проверить более сложные свойства, следует использовать всю мощь выражения интерфейса Python. Это будет предметом будущего Gem.

Связанный со статьёй текст программы

File: test.cmd

```
gnatmake -g p.adb
gdb p --command=scenario.gdb
```

File: scenario.gdb

```
set variable $account_open := False

break open_account
commands
 if $account_open = True
  echo error: account opened twice\n
 else
  set variable $account_open := True
  printf "(info) account created with %d$\n", cash
  continue
 end
end

break deposit
commands
 if $account_open = False
  echo error: someone tries to deposit before opening an account\n
 else
  printf "(info) deposit: %d$\n", cash
  continue
 end
end

break withdraw
commands
 if $account_open = False
  echo error: someone tries to withdraw before opening an account\n
 else
  printf "(info) withdrawal: %d$\n", cash
  continue
 end
end

run
```

File: p.adb

```
1. with Bank;
2.
3. procedure P is
4. begin
5.    Bank.Open_Account (100);
6.    Bank.Deposit (45);
7.    Bank.Withdraw (55);
8.    Bank.Deposit (80);
9. end P;
```

File: bank.ads

```
1. package Bank is
2.
3.    procedure Open_Account (Cash : Integer);
4.
5.    procedure Deposit (Cash : Integer);
6.
7.    procedure Withdraw (Cash : Integer);
8.
```

```
9. end Bank;
```

File: bank.adb

```
1. package body Bank is
 2.
 3.    Balance : Integer;
 4.
 5.    procedure Open_Account (Cash : Integer) is
 6.    begin
 7.       Balance := Cash;
 8.    end Open_Account;
 9.
10.    procedure Deposit (Cash : Integer) is
11.    begin
12.       Balance := Balance + Cash;
13.    end Deposit;
14.
15.    procedure Withdraw (Cash : Integer) is
16.    begin
17.       Balance := Balance - Cash;
18.    end Withdraw;
19.
20.    -- To celebrate the inauguration of the bank at elaboration time, your
21.    -- account is credited with 100$.
22.
23.    Special_Offer : constant Integer := 100;
24. begin
25.    Deposit (Special_Offer);
26. end Bank;
```

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Jerome присоединился к AdaCore в 2002 после завершения его исследований в Ecole Nationale des Télécommunations (Париж, Франция), во время которого он уже работал с компанией на одном из ее многих reseach проектов, а именно, PolyORB. Его энтузиазм остался непотускневшим в течение этих шести лет, и он работал над множеством проектов, а также экспертным знанием получения в отладчиках и перекрестных технологиях. Он недавно приложил усилия к порту GNAT Pro в различных перекрестным платформам (в частности ОС VxWorks 6, ELinOS).

Last Updated: 11/24/2017
Posted on: 3/12/2012

Обсуждение…