Gem #151: Определение математических параметров программы

Автор: Yannick Moy — AdaCore

Краткое содержание: Добавление множества новых утверждений в Ada 2012 может вызвать желание утвердить параметры данных, которые не учитывают возможность возникновения переполнений. В GNAT определены функция переключения и прагма, с помощью которых можно добиться именно такого эффекта.

Давайте начнём…

Целочисленные переполнения - это экзотические и опасные звери, с которыми большинство программистов сталкиваются не очень часто и о которых обычно забывают. Целочисленное переполнение происходит, когда результат арифметического вычисления не помещается в целочисленный тип машины, который должен содержать результат. Конечно, Ada требует проверки во время выполнения для защиты от целочисленных переполнений, которые включаются ключом `-gnato` в GNAT. Но для производственных бинарных кодов без компиляции обычно происходит компиляция, и в этом случае целочисленное переполнение приведет к тому, что в Справочном руководстве Ada называется «ошибочное поведение», что означает, что может произойти все что угодно (см. Gems # 132 - # 135).

Давайте рассмотрим функцию `Max_Payload`, вычисляющую максимальную полезную нагрузку меньше емкости `Capacity`, которую можно построить с помощью двух элементов `It1` и `It2`:

```
package Pack is

   type Payload is new Natural;

   function Max_Payload
     (It1, It2 : Payload;
      Capacity : Payload) return Payload;

end Pack;
```

Реализация Max_Payload пытается сначала соответствовать самой большой полезной нагрузке, а затем самой маленькой:

```
package body Pack is

   function Max_Payload
     (It1, It2 : Payload;
      Capacity : Payload) return Payload
   is
      Result : Payload := 0;
      Small  : Payload := Payload'Min (It1, It2);
      Big    : Payload := Payload'Max (It1, It2);
   begin
      if Big <= Capacity then
         Result := Big;
      end if;

      if Small <= Capacity - Result then
         Result := Result + Small;
      end if;
```

```
      return Result;
   end Max_Payload;

end Pack;
```

Обратите внимание, что тест:

```
      if Small <= Capacity - Result then
```

написан таким образом, чтобы избежать целочисленных переполнений, в то время как более естественный способ написания этого теста:

```
      if Small + Result <= Capacity then  --  incorrect
```

уязвим для переполнения целых чисел, если `Small + Result` больше максимального целого числа.

Хотя предполагается писать такие неестественные выражения в коде, чтобы избежать переполнения целых чисел, мы хотели бы написать спецификации (например, контракты подпрограмм) более математическим способом. Например, естественным способом выражения `postcondition` для функции `Max_Payload` является:

```
function Max_Payload
  (It1, It2 : Payload;
   Capacity : Payload) return Payload;
  with Post =>
    Max_Payload'Result =
      (if It1 + It2 <= Capacity then It1 + It2
       elsif It1 <= Capacity and (It1 >= It2 or It2 > Capacity) then It1
       elsif It2 <= Capacity then It2
       else 0);
```

Поскольку контракты выполняются в Ada, их можно скомпилировать как утверждения времени выполнения при передаче ключа `-gnata` в GNAT. (Для более точного контроля выполнения утверждений см. Gem # 149.)

Давайте проверим приведенную выше реализацию:

```
with Pack; use Pack;

procedure Test_Pack is
begin
   pragma Assert (Max_Payload (1, Payload'Last, 10) = 1);
end Test_Pack;
```

Компиляция и запуск приводят к ошибке во время выполнения, потому что `It1 + It2` не помещается в целое число:

```
$ gnatmake -gnata -gnato test_pack.adb
$ ./test_pack

raised CONSTRAINT_ERROR : pack.ads:10 overflow check failed
```

Означает ли это, что мы не можем написать спецификации наиболее естественным образом? В GNAT ответ «нет», используя альтернативный механизм проверки переполнения для утверждений

(включая контракты подпрограмм, прагма `Assert` и т. д.)Идея состоит в том, чтобы использовать 64-битные целые числа (`Long_Long_Integer`) для арифметических вычислений в утверждениях, чтобы исключить возможность переполнения в большинстве случаев. Это может быть достигнуто либо путем компиляции с ключом `-gnato12`, либо путем добавления следующей прагмы в `pack.adb` или в файл конфигурации:

```
pragma Overflow_Mode (General => Strict, Assertions => Minimized);
```

Компиляция и запуск сейчас не приводят к ошибкам:

```
$ gnatmake -gnata -gnato12 -s test_pack.adb
$ ./test_pack
```

Обратите внимание, что GNAT использует 64-разрядные целые числа только тогда, когда они необходимы, основываясь на знании границ статического типа. Другой режим (`Eliminated`, также запускаемый с ключом `-gnato13`) предписывает компилятору полностью исключить возможность переполнения с помощью библиотеки времени выполнения целых чисел с бесконечной точностью. Наконец, альтернативные режимы переполнения также могут использоваться для кода, а также для утверждений, если пользователь желает. Для получения дополнительной информации о режимах переполнения см. Руководство пользователя GNAT.

PS: Все еще не уверены, что тело `Max_Payload` реализует свой контракт? Поскольку приведенный выше код написан в `SPARK 2014`, просто используйте инструмент `GNATprove`, чтобы доказать это! Это то, что я сделал.

Связанный со статьёй текст программы

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

Об авторе

Yannick Moy — AdaCore

Работа Yannick Moy сосредоточена на анализе исходного кода программного обеспечения, в основном для обнаружения ошибок или проверки свойств безопасности / обеспечения безопасности. Yannick ранее работал в PolySpace (теперь The MathWorks), где он начал проект C ++ Verifier. Затем он поступил в INRIA Research Labs / Orange Labs во Францию для проведения PhD по автоматической модульной проверке статической безопасности для программ на C. Янник присоединился к AdaCore в 2009 году, после короткой стажировки в Microsoft Research.

Yannick Moy имеет степень инженера из Политехнической школы (Ecole Polytechnique), магистра в Стэнфордском университете (Stanford University) и доктора философии из Университета Парижа-Суд (Université Paris-Sud). Он является учёным Siebel.

Last Updated: 10/13/2017
Posted on: 7/1/2013

Обсуждение…