## *Gem #147: Подтипы в Ада 2012 - Часть 2*

**Автор:** Yannick Moy — AdaCore

Краткое содержание: В предыдущем Gem серии мы показали, как использовать `Static_Predicate` для определения неизменяемых параметров скалярных объектов. Тема данного Gem - аспект `Dynamic_Predicate`.

**Давайте начнём…**

Предыдущий Gem в этой серии показал, как аспект `Static_Predicate` может использоваться для определения свойств скалярных объектов, которые должны соблюдаться всегда. Этот Gem связан с аспектом `Dynamic_Predicate`, который можно использовать во всех объявлениях типов и подтипов (не только в скалярных).

Рассмотрим, например, тип `Message`, кодирующий даты, когда сообщение было отправлено и получено, где даты представлены строками, такими как «`1789-07-14`» для четырнадцатого июля 1789 года:

```
type Day is new String (1 .. 10);

type Message is record
   Sent     : Day;
   Received : Day;
end record;
```

Чтобы указать, что дата получения сообщения всегда должна быть больше даты его отправки, мы можем написать:

```
type Message is record
   Sent     : Day;
   Received : Day;
end record with
  Dynamic_Predicate => Message.Sent <= Message.Received;
```

Обратите внимание, что само имя типа используется в качестве префикса компонентов, названных в предикате. В этом контексте имя типа обозначает то, что Ada называет текущим экземпляром типа, который во время выполнения будет обозначать фактический объект, к которому применяется предикат.

В отличие от `Static_Predicate`, компилятор не может определить в целом, произойдет ли сбой `Dynamic_Predicate`, поэтому он вставляет проверки во время выполнения в определенные требуемые местоположения в коде:

- при назначении переменной подтипа
- при передаче входного параметра подтипа
- при возврате выходного параметра объекта подтип
- при преобразовании значения в подтип

Например, на следующий неправильный код:

```
M : Message := (Received => "1776-07-04", Sent => "1783-09-03");  --  incorrect
```

Компиляция с утверждениями и запуск приводит к следующей ошибке:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : Dynamic_Predicate failed at main.adb:3
```

Если значения отправленного и полученного компонентов скорректированы, чтобы отразить фактический порядок событий провозглашения независимости Соединенных Штатов и дату заключения Парижского договора, завершающего американскую революционную войну, то сгенерированный код выполняется без ошибок.

Помните, что при назначении отдельных компонентов проверки выполнения не вставляются, поэтому предикат может быть молчаливо нарушен между назначениями и вызовами. Например, если приведенное выше определение отдельно присваивает каждому компоненту `M`, даже если значения для `Received` и `Sent` упорядочены соответствующим образом:

```
   M : Message;  --  incorrect
begin
   M.Received := "1783-09-03";  --  incorrect
   M.Sent := "1776-07-04";      --  predicate is correct here
```

Этот код не приводит к отказу во время выполнения, но если мы передаем сообщение, прежде чем это будет полностью инициализировано к некоторому `Process` процедуры, берущему его в качестве входного параметра:

```
procedure Process (M : Message);
```

Компиляция получающегося кода с утверждениями и выполнение его снова приводят к ошибке:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : Dynamic_Predicate failed at main.adb:7
```

Обратите внимание, что `Dynamic_Predicate` является более гибким, чем `Static_Predicate`: он может применяться к большему количеству форм типов и более общих выражений предикатов. Например, оператор `mod` не допускается вне статического выражения в `Static_Predicate`, поэтому тип нечетных чисел должен быть определен с помощью `Dynamic_Predicate`:

```
subtype Odd is Integer with Dynamic_Predicate => Odd mod 2 = 1;
```

Аналогично, пользовательская функция может вызываться в `Dynamic_Predicate`, но не в `Static_Predicate`.

GNAT, для удобства, предоставляет аспект `Predicate`, который можно использовать везде, где разрешен `Dynamic_Predicate`, и анализирует его как `Static_Predicate`, когда это возможно.

В следующем и последнем Gem этой серии о контрактах типов и подтипов мы рассмотрим связанный аспект `Type_Invariant`.

**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: 5/6/2013*

**Обсуждение…**