Gem #124: Скрипты GPS для статического анализа

Авторы: Yannick Moy, Nicolas Setton - AdaCore

Краткое содержание: Возможность осуществления быстрого запроса простых параметров из базы кода ценна как в ходе разработки, так и на этапе отладки. В большинстве случаев можно обойтись простыми решениями, наподобие `grep`, или обычным скриптом, основанным на запросах синтаксиса. Для более сложных запросов, требующих знания семантики, необходимо применять доступные инструменты, которые не всегда идеально подходят.

В данном Gem#124 описывается способ осуществления запросов с помощью возможностей скриптования GPS на Python - IDE GNAT Programming studio..

Давайте начнём…

Возможность быстро запрашивать базу кода для некоторых простых свойств является ценной возможностью, как при разработке, так и при отладке. Большую часть времени можно обойтись грубыми запросами типа `grep` или простым скриптом, основанным на синтаксических запросах. Для более сложных запросов, требующих семантических знаний, необходимо использовать существующие инструменты, которые не всегда идеально подходят.

Этот Gem#124 обеспечивает способ выполнения этих запросов, используя возможности сценариев Python GPS, IDE студии программирования GNAT.

В качестве примера рассмотрим критический раздел, API которого определяет процедуры Enter и Leave:

```
package Critical_Section is

    procedure Enter;
    --  Enter the critical section

    procedure Leave;
    --  Leave the critical section.
    --  Important: every procedure calling Enter should also call Leave.

end Critical_Section;
```

Для иллюстрации их реализация может быть такой же простой, как:

```
package body Critical_Section is

    In_Critical : Boolean := False;

    procedure Enter is
    begin
       In_Critical := True;
    end Enter;

    procedure Leave is
    begin
       In_Critical := False;
    end Leave;

end Critical_Section;
```

Эти критические разделы могут использоваться в основной программе более или менее корректными способами:

```
with Critical_Section; use Critical_Section;

procedure Main is

   procedure A is
   begin
      Enter; -- Do some processing Leave;
   end A;

   procedure B is
   begin
      Enter; -- Do some processing
   end B;

begin
   A;
   B;
end Main;
```

В идеале мы хотели бы, чтобы каждая подпрограмма, которая входит в критический раздел, также оставляла его. Это требует знания графа вызовов приложения, а не того, что может решить групповой подход, и вряд ли у вас будет инструмент, который уже вычисляет эту информацию для вас.

Ну, это очень легко сделать в GPS!

Откройте GPS на проекте и скомпилируйте код с помощью GNAT. Это создает кросс-справочной информации, сохраненной в .файлы ali, содержащие, в частности, ссылки на все сущности в коде. Python API в GPS позволяет запрашивать эту информацию, знать, где каждая сущность ссылается, определить вид сущности и т.д.

Откройте оболочку Python с помощью Window->Python и введите следующие строки в оболочку (кроме комментариев, введенных #):

```
# get the entities corresponding to the functions
enter_entity = GPS.Entity("Enter", GPS.File("critical_section.ads"))
leave_entity = GPS.Entity("Leave", GPS.File("critical_section.ads"))

# get the list of calls for each function

enter_called_by = enter_entity.called_by()
leave_called_by = leave_entity.called_by()

# print the result
print ["Enter is called by %s, which does not call Leave" % fun for fun
in enter_called_by if not fun in leave_called_by]
```

Это должно вывести следующий результат:

```
['Enter is called by B:main.adb:12:14, which does not call Leave']
```

Это информация, которую мы искали!

Теперь, вы можете найти этот вид запроса достаточно распространенным, что вы хотели бы иметь простую кнопку, чтобы нажать в GPS, чтобы вычислить этот результат. Возможности плагина GPS позволяют зарегистрировать код выше, так что пользователю будет предложено ввести и оставить две подпрограммы. Сам плагин просто:

```
""" Given the names of two subprograms Enter and Leave defining a
critical
     section, this plug-in detects subprograms which only enter the
critical
     section without leaving it.
"""

import GPS

def detect_critical_section_violation(self):
     enter_name, enter_file, leave_name, leave_file=
GPS.MDI.input_dialog("Enter the names of subprograms defining the
critical section:", "Enter", "File containing Enter", "Leave", "File
containing Leave")


     # get the entities corresponding to the functions
     enter_entity = GPS.Entity(enter_name, GPS.File(enter_file))
     leave_entity = GPS.Entity(leave_name, GPS.File(leave_file))

     # get the list of calls for each function

     enter_called_by = enter_entity.called_by()
     leave_called_by = leave_entity.called_by()

     # compute the violations
     violations = [fun for fun in enter_called_by if not fun in
leave_called_by]
     if violations == []:
         GPS.MDI.dialog ("No violation found")
     else:
         GPS.MDI.dialog(str(["Enter is called by %s, which does not call
Leave" % fun for fun in violations]))


def on_gps_started (hook_name):
     menu = GPS.Menu.create("/Tools/Browsers/Detect critical section
violation", detect_critical_section_violation)


GPS.Hook ("gps_started").add (on_gps_started)
```

Поместите этот код в файл critical_section.py под любой из `INSTALL/share/gps/plug-ins` или `HOME/.gps/plug-ins` и меню будут там в следующий раз, когда вы откроете GPS.

Для получения дополнительной информации о том, как использовать Python API в GPS, мы ссылаемся на раздел 16 GPS User's Guide (Customizing and Extending GPS) и Python API доступны в разделе Help->Python extensions in GPS.

Связанный со статьёй текст программы

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

Об авторах

Соавтор Nicolas Setton

Опыт в AdaCore (по доступным материалам LinkedIn):

Ответственный за GPS, флагман IDE для AdaCore:

Решение и реализация стратегической дорожной карты для развития IDE, координация технических усилий и позиционирования маркетинга.

Ответственный за порт GNAT Pro для Mac OS: Координация технических усилий и обработка переходов на новые версии ОС.

Ответственный за дополнение GtkAda: Поддержание стратегической дорожной карты для этого продукта и координация технических усилий.

Создана инфраструктура виртуализации в парижском машинном зале. Часть команды, которая поддерживает 25 пользователей и 40 систем в различных ОС.Работа над графическими пользовательскими интерфейсами, как часть команды разработчиков, работающих над IDE.

Работа над компилятором: отвечает за перенос компилятора GNAT на Mac OS X на процессорах PowerPC и x86.Реализованы различные улучшения генерации отладочной информации.

По инфраструктуре тестирования и качества: Создано приложение для контроля качества программного обеспечения и формализм для выражения планов тестирования и обеспечения качества для всех программных продуктов компании. Этот инструмент в настоящее время находится в производстве и занимает центральное место в гибкой инфраструктуре квалификации и выпуска на AdaCore.

Образование:

Национальная высшая школа связи / Telecom Paris

Наименование степени Diplôme d'Ingénieur (степень магистра в области машиностроения)

Область исследования Информатика

Автор Yannick Moy

Работа Yannick Moy сосредоточена на анализе исходного кода программного обеспечения, в основном для выявления ошибок или проверки свойств безопасности. Янник ранее работал в PolySpace (теперь The MathWorks), где он начал проект C ++ Verifier. Затем он присоединился к исследовательской лаборатории INRIA / Orange Labs во Франции, чтобы получить степень доктора наук по автоматической модульной проверке статической безопасности для программ на языке Си. Янник присоединился к AdaCore в 2009 году, после небольшой стажировки в Microsoft Research.

Янник получил степень инженера в Политехнической школе, степень магистра в Стэнфордском университете и степень доктора в Université Paris-Sud. Он Siebel Scholar.

Last Updated: 10/13/2017
Posted on: 4/30/2012

Обсуждение…