Gem #134: Ошибочное выполнение - Часть 3

Автор: Bob Duff—AdaCore

Краткое содержание: В данном Gem продолжается обсуждение примера ошибочного выполнения, начатого в части 2.

Давайте начнём…

Мы показали, как этот код:

```
X : Natural := ...;
Y : Integer := X + 1;
if Y > 0 then
    Put_Line (Y'Img);
end if;
```

может закончить тем, что печатал отрицательное число, если проверки подавлены.

Фактически, хороший компилятор предупредит, что «`Y> 0`» обязательно True, поэтому код глуп, и вы можете это исправить. Но вы не можете рассчитывать на это. Оптимизаторы способны на гораздо более тонкие рассуждения, которые могут не выдавать предупреждение. Например, предположим, у нас есть процедура:

```
procedure Print_If_Positive (Y : Integer) is
begin
    if Y > 0 then
        Put_Line (Y'Img);
    end if;
end Print_If_Positive;
```

Кажется «очевидным», что `Print_If_Positive` никогда не будет печатать отрицательное число. Но при наличии ошибочности (*прим.* и подавлении проверок при исполнении кода) эти рассуждения не работают:

```
X : Natural := ...;
Y : Integer := X + 1;
Print_If_Positive (Y);
```

Оптимизатор может решить встроить вызов (*прим.* типа on-line procedure), а затем оптимизировать, как в предыдущем примере.

Другие языковые функции, которые могут вызвать ошибочное выполнение, включают:

- Общие переменные (ошибочно, если не удается синхронизировать несколько задач)
- Адреса пунктов
- Unchecked_Conversion
- Интерфейс к другим языкам
- Вставки машинного-кода
- Пользовательские пулы хранения
- Unchecked_Deallocation

Полный список можно найти, посмотрев "erroneous" ("ошибочный") в индексе RM или выполнив поиск в RM. Каждый случай ошибочного исполнения задокументирован под заголовком "Erroneous Execution" ("ошибочное исполнение").

Вы должны попытаться свести к минимуму использование таких функций. Когда вам нужно использовать их, попробуйте инкапсулировать их, чтобы вы могли рассуждать о них локально. И будьте осторожны.

Что касается подавления проверок: не подавляйте, если вам не нужна дополнительная эффективность, и у вас есть уверенность, что проверки не потерпят неудачу. Если вы подавляете проверки, регулярно выполняйте регрессионные тесты в обоих режимах (подавлено и не подавлено).

Заключительная часть этой серии Gem объяснит обоснование концепции ошибочного выполнения в Ada.

Связанный со статьёй текст программы

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

Об авторе

Автор: Bob Duff

AdaCore

Роберт (Боб) Андерсон Дафф - дизайнер языков программирования. Написал основные части Ada Language Reference Manual. При его участии разработаны и другие языки. Изучил десятки языков программирования.

Разработал, внедрил и/или внес большой вклад в десять компиляторов, три инструмента статического анализа и различные программные системы от реального времени/встроенные до параллельных и распределенных систем.

Эксперт в ведении программных проектов для своевременного их завершения.

Опыт работы в AdaCore:

Senior Software Engineer
AdaCore
2005 – настоящее время 14 лет
New York and Massachusetts

Ранее:

SofCheck, Inc,
Intermetrics,
Oak Tree Software, Inc.

Образование:

Carnegie Mellon University
Bachelor of Science (BS), Applied Mathematics with concentration in Computer Science, 3.6/4.0.
1982
Деятельность и сообщества: Explorer's Club

Last Updated: 10/13/2017
Posted on: 9/10/2012

Обсуждение…