Gem #25: Как осуществлять поиск текста

Автор: Emmanuel Briot, Senior Software Engineer, AdaCore

Краткое содержание: Gem Ada #25 - Стандартом Ada определяется несколько подпрограмм, связанных с поиском текста в строке. Кроме того, в GNAT доступны дополнительные пакеты поиска. В данном «gem» описываются эти функции.

Давайте начнем…

Стандарт Ada определяет несколько подпрограмм, связанных с поиском текста в строке. Кроме того, gnat предоставляет дополнительные пакеты для поиска. Gem #25 будет охватывать различные возможности.

Мы предполагаем, что ищем текст в последовательности символов, которая находится в памяти. Чтобы искать текст в файле, самое простое это загрузить файл в память, и затем искать получающуюся последовательность.

Мы также предполагаем, что мы имеем дело со строковым типом String, а не с Unbounded_String или Bounded_String. В обоих случаях можно легко преобразовать в строку. Предустановленные подпрограммы Ada существуют для всех трех этих типов, но специфические пакеты GNAT имеют дело только со строками. Конечно, преобразование в строку не обязательно эффективно, и нужно указывать на пакеты, специфичные для GNAT, Ada.Strings.Unbounded.Aux, которые позволяют заглянуть внутрь неограниченной строки, но, очевидно, следует использовать их с осторожностью.

Текст, который мы ищем (образец или шаблон) может иметь различные формы. Это может быть фиксированный текст, который должен быть найден точно так же, как есть, или это может быть регулярное выражение, которое соответствует целому набору строк.

При поиске фиксированного текста очевидными кандидатами являются предопределенные подпрограммы времени выполнения Ada из пакета Ada.Strings.Fixed. Соответствующие подпрограммы начинаются с префикса Index. Они разумно оптимизированы, хотя они не написаны специально для сборки для максимальной скорости. Аналогичные подпрограммы существуют для ограниченных и неограниченных строк.

Для поиска фиксированного текста в длинных текстах существуют некоторые расширенные алгоритмы. Эти алгоритмы проводят некоторое время, изучая шаблон изначально, создавая внутреннее представление (например, конечное состояние автомата), с тем чтобы последующие поиски стали более эффективными. Одним из примеров этого является алгоритм Бойера-Мура. Начальные вычисления могут быть более дорогостоящими, чем время, затрачиваемое на поиск, когда строка невелика, поэтому использование индекса в таких случаях является наиболее эффективным решением. Однако по мере роста размера строки, использование более продвинутого алгоритма становится интересным. Среда выполнения gnat в настоящее время не предоставляет такую реализацию, но среда разработки GPS имеет такой пакет, который, вероятно, будет интегрирован в среду выполнения gnat в будущем. Сообщите нам о Вашей заинтересованности в таком пакете!

Регулярные выражения являются хорошо известной функцией на нескольких языках программирования, таких как Perl, и предоставляют расширенные возможности

поиска. Они предоставляют синтаксис для описания шаблона, который должен быть сопоставлен. Например, «a.*b» найдет все подстроки, начинающиеся с «a» и заканчивающиеся на «b», с нулем или более символами между ними. Аналогично, «a [bc] d» найдет все трехсимвольные подстроки, начинающиеся с «a», заканчивающиеся на «d», а средний символ - «b» или «c». Этот камень не является учебником о том, как писать регулярные выражения, поскольку, например, руководство по Perl или различные учебные пособия в Интернете уже подробно освещают тему.

GNAT предоставляет два пакета для поиска регулярных выражений.

Самый известный из них - GNAT.Regexp. Это не полностью общий пакет регулярных выражений, поскольку он не поддерживает, например, получение подстрок, согласованных группами скобок. Он также пытается сопоставить весь текст; то есть «a [bc] d» будет соответствовать только, если сам текст имеет три символа. Он поддерживает две версии регулярных выражений, обычные, как описано выше, и то, что обычно называют шаблонами «globbing», которые используются различными командами командной строки в Unix. Поэтому GNAT.Regexp чаще всего используется для поиска, связанного с именами файлов.

Вот короткий пример о том, как найти все исходные файлы Ada в текущем каталоге. Как Вы заметите, регулярное выражение сначала должно быть скомпилировано в конечный автомат, и затем обеспечивает очень эффективное сопоставление с образцом через функцию Соответствия – Match.

```
with Ada.Directories; use Ada.Directories;
with GNAT.Regexp;     use GNAT.Regexp;

procedure Search_Files (Pattern : String) is
  Search : Search_Type;
  Ent    : Directory_Entry_Type;
  Re     : constant Regexp := Compile (Pattern, Glob => True);

begin
  Start_Search (Search, Directory => ".", Pattern => "");
  while More_Entries (Search) loop
     Get_Next_Entry (Search, Ent);
     if Match (Simply_Name (Ent), Re) then
         ... --  Matched, do whatever
     end if;
  end loop;
  End_Search (Search);
end Search_Files;

Search_Files ("*.adb");
```

Второй пакет, который обеспечивает GNAT, является GNAT.Regpat. Этот пакет обеспечивает полную обработку обычных конструкций регулярного выражения, но не некоторые недавние усовершенствования, которые языки такие как Perl уже используют (предвидение – look-ahead, unicode, …). Регулярное выражение сначала скомпилировано в байт-код, который обеспечит более быстрое соответствие. Обратите внимание на то, что этот пакет не так эффективен как GNAT.Regexp, так как некоторые образцы считают символы, обнаружат, что они не соответствуют, затем отслеживают в обратном порядке к более ранней позиции и пробуют некоторую другую возможность. На практике эффективность обычно достаточно хороша.

Одна из возможностей, которую этот пакет обеспечивает над GNAT.Regexp, - это возможность после сопоставления узнать, какие группы скобок совпадают. Например, в

регулярном выражении «a (большое | маленькое) приложение», можно узнать, какая из двух альтернатив была сопоставлена, как в следующем примере:

```
with GNAT.Regpat;   use GNAT.Regpat;
with Ada.Text_IO;   use Ada.Text_IO;

procedure Search_Regexp (Pattern : String; Search_In : String) is
  Re      : constant Pattern_Matcher := Compile (Pattern);
  Matches : Match_Array (0 .. 1);

begin
  Match (Re, Search_In, Matches);
  if Matches (0) = No_Match then
     Put_Line ("The pattern did not match");
  else
     Put_Line ("The application really is "
               & Search_In (Matches (1).First .. Matches (1).Last));
  end if;
end Search_Regexp;

Search_Regexp ("a (big|small) application",
               "Example of a small application for searching");
```

Несмотря на то, что регулярные выражения довольно мощны, и обычно более чем достаточно для большинства приложений, у них есть ограничения, которые может преодолеть только полная бесконтекстная грамматика. Один классический пример - то, что они не могут гарантировать, что у последовательности есть то же количество открывающих и закрывающих скобок, как в “(a(b))”, когда количество круглых скобок не известно заранее.

GNAT обеспечивает пакет GNAT.Spitbol.Patterns, который содержит различные подпрограммы, которые могут использоваться, чтобы реализовать соответствие с бесконтекстными грамматиками. Сам SPITBOL - полный язык программирования, который GNAT может эмулировать через подпрограммы в GNAT.Spitbol, но в Gem #25 мы только концентрируемся на возможностях сопоставления с образцом.

Вот небольшой пример использования GNAT.Spitbol.Patterns. Цель здесь состоит в том, чтобы проверить, начинается ли заданная строка со сбалансированным выражением относительно «[» и «{», то есть если первый символ является одним из них, тогда должен быть аналогичный замыкающий символ, а подстрока в между ними также должным образом сбалансированы.

Язык для такого образца в формате BNF был бы:

```
  ELEMENT ::= <any character other than [] or {}>
              | '[' BALANCED_STRING ']'
              | '{' BALANCED_STRING '}'
  BALANCED_STRING ::= ELEMENT {ELEMENT}
```

который переводится в следующую программу spitbol (см. документацию в GNAT.Spitbol.Patterns для получения дополнительной информации о формате шаблонов). Обратите внимание, что эта программа не заставляет конец строки правильно сбалансироваться, а только ее начало.

```
with GNAT.Spitbol.Patterns;  use GNAT.Spitbol.Patterns;

procedure Search_Spitbol is
  Element, Balanced_String : aliased Pattern;
  At_Start : Pattern;
```

```
begin
  Element := NotAny ("[]{}")
        or ('[' & (+Balanced_String) & ']')
        or ('{' & (+Balanced_String) & '}');
  Balanced_String := Element & Arbno (Element);
  At_Start := Pos (0) & Balanced_String;
  Match ("[ab{cd}]",   At_Start); --  True
  Match ("[ab{cd}",    At_Start); --  False
  Match ("ab{cd}",     At_Start); --  True
end Search_Spitbol;
```

Связанный исходный код

Attached Files

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

Обсуждение…

2 Комментария

Leo Brewin
24-го мая 2013
Привет Emmanuel,
Спасибо за примеры.
Я думаю, что во втором примере есть небольшая ошибка.
Слово "Simply" в строке
if Match (Simply_Name (Ent), Re) then
должно, вероятно, быть “ Simple”. Я думаю, что корректная строка должна быть
if Match (Simple_Name (Ent), Re) then
С наилучшими пожеланиями,
Leo Brewin

Gary Dismukes
24-го мая 2013

Leo,
Да, это должно быть Simple_Name. Спасибо за исправление, теперь исправлено!
Gary Dismukes