The Design of the Byron Translator

Overview

Translating a program can be viewed at a high-level as a function with the following profile:

`Translate( Source : Source_Type ) return Target_Type`

which itself can be seen as being a composed function, where each of the normal phases of the compiler are the composing functions such that:

```
Function Translate( Source : Source_Type ) return Target_Type is
  ( Generation(Semantic_Analyzer(Parser( Lexer(Source)) )) );
```

To facilitate this compositional construction, Byron defines several generics — such as the following:

```
Pragma Ada_2012;
Pragma Assertion_Policy( Check );

-- Byron.Pass is a generic function intended to encapsulate the idea of a
-- compiler pass; it does this by applying a translation -- transformations
-- are applied to the input (and the output), if given.
Generic
   Type Input_Type(<>)  is private;
   Type Output_Type(<>) is private;
   with Function "="(Left, Right : Input_Type ) return Boolean is <>;
   with Function "="(Left, Right : Output_Type) return Boolean is <>;
   with Function Translate(Input : Input_Type) return Output_Type;
   Input_Transformation  : access Procedure( Object : in out Input_Type  ):= Null;
   Output_Transformation : access Procedure( Object : in out Output_Type ):= Null;
Function Byron.Generics.Pass( Input : Input_Type ) Return Output_Type;
```

— the above defines the generic function corresponding to a pass in the compiler. As examples the Lexer is defined as an instantiation taking a string (containing the whole compilation-unit) and returning a sequence of tokens, the Reader is an instantiation taking a file-stream and returns the contents as a string and thus tokenizing from a file can be accomplished by calling `Byron.Lexer( Byron.Reader(Stream) ).`