Easy Ada tooling with Libadalang

Pierre-Marie de Rodat Raphaël Amiard

Software Engineers at AdaCore

The need

In three bullet points

- A library that allows user to query data about Ada programs/libraries.
- Syntactic and semantic information
- Both low level and high level information
- Should be easy to integrate into tools/IDEs from different platforms/languages/ect.

The need - IDEs

```
174     end record;
175
176     type Cond_Branch_Context is limited record
177        Decision_Stack : Decision_Occurrence_Vectors.
178        --  The stack of open decision occurrences
179
180        Basic_Blocks   : Basic_Block_Sets.Set;
181        --  All basic blocks in the routine being ana
182
183        Stats          : Branch_Statistics;
184        --  Statistics on conditional branches in the
185
186        Subprg         : Address_Info_Acc;
187        --  Info of enclosing subprogram
188     end record;
189
190     procedure Analyze_Routine
191       (Name   : String_Access;
```

Figure 1: Syntax & block highlighting

The need - IDEs

```
 8   function "+"
 9     (S1, S2 : Unbounded_String)
10      return Unbounded_String is
11     (S1 & "." & S2);
12
13   function Concat (Ns : String_Array) return String
14   is
15      R : Unbounded_String;
16   begin
17      for N of Ns loop
18         if Length (R) = 0 then
19            R := N;
20         else
21            R := R & N;
22         end if;
23      end loop;
24   end Concat;
```

```
 8   function Concat
 9     (S1, S2 : Unbounded_String)
10      return Unbounded_String is
11     (S1 & "." & S2);
12
13   function Concat (Ns : String_Array) return String
14   is
15      R : Unbounded_String;
16   begin
17      for N of Ns loop
18         if Length (R) = 0 then
19            R := N;
20         else
21            R := Concat (R, N);
22         end if;
23      end loop;
24   end Concat;
```

Figure 2: Refactoring

The need - IDEs

Figure 3: Cross references

The need - command line tools

```
procedure Main is
   type my_int is new Integer range 1 .. 10;
   Var : my_int := 12;
begin
   null;
end Main;
```

```
$ ./my_custom_lal_checker main.adb

main.adb:2:9: Variable should start with lowercase letter
main.adb:3:4: Type name should start with uppercase letter
```

Why not ASIS/GNAT?

Challenges

- Incremental: Don't recompute everything when the code changes
- Error recovery: Ability to compute partial results on incorrect code
- Long running: Be able to run for 3 days without crashing your machine

GNAT and AdaCore's ASIS implementation are ill suited to those challenges.

Design goals

- Multi-language: Easy binding generation to other languages/ecosystems
  - Today: Python, Ada, C
- Easy scripting: Be able to create a prototype quickly & interactively
- Both low & high level APIS:
  - What type is this expression?
  - How many references to this variable?
  - Give me the source location of this token
  - Rename this entity
  - Etc.

API Part 1: Tokens

```
ctx = lal.AnalysisContext()
unit = ctx.get_from_file('main.adb')
for token in unit.root.tokens:
    print('Token: {}'.format(token))
```

API Part 2: Syntax

```
for object_decl in unit.root.findall(lal.ObjectDecl):
    print(object_decl.text)
```

API Part 3: Semantic

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is
    function Double (I : Integer) return Integer is (I * 2);
    function Double (I : Float) return Float is (I * 2.0);
begin
    Put_Line (Integer'Image (Double (12)));
end Main;
```

```
double_call = unit.root.find(
    lambda n: n.is_a(lal.CallExpr) and n.f_name.text == 'Double'
)

print(double_call.f_name.p_referenced_decl)
```

API Part 4: Tree rewriting

WARNING: Not done yet

```
diff = ctx.start_rewriting()

# Get the first parameter of the call to Double
param_diff = diff.get_node(double_call.f_suffix[0])

# Replace the expression of the parameter with a new node
param_diff.f_expr = lal.rewriting.RealLiteral('12.0')

diff.apply()
```

An example

Program that checks some property on nodes (e.g. variables names must be capitalized).

Technical prototypes/demos

Syntax highlighter/Xref explorer

```
with Ada.Text_IO; use Ada.Text_IO;
use all type Ada.Text_IO.File_Type;

procedure Example is

   subtype Nat is Integer range 0 .. Integer'Last;

   type Rec (N : Natural) is tagged record
      S : String (1 .. N);
   end record;

   type Money_Type is delta 0.01 digits 14;

   generic
      with procedure Put_Line (S : String);
   package Things is
      procedure Process (S : access Wide_String)
        with Pre => S /= null and then S'Length > 0
                    and then (for all I in S.all'Range =>
                                S.all (I) / ASCII.NUL);
   end Things;

   package body Things is

      -------------
      -- Process --
      -------------

      procedure Process (S : access Wide_String) is
```

Figure 4: Libadalang based highlighter

Syntax based static analyzers

```
def has_same_operands(binop):
    def same_tokens(left, right):
        return len(left) == len(right) and all(
            le.is_equivalent(ri) for le, ri in zip(left, right)
        )
    return same_tokens(list(binop.f_left.tokens), list(binop.f_right.tokens))

def interesting_oper(op):
    return not op.is_a(lal.OpMult, lal.OpPlus, lal.OpDoubleDot,
                       lal.OpPow, lal.OpConcat))

for b in unit.root.findall(lal.BinOp):
    if interesting_oper(b.f_op) and has_same_operands(b):
        print 'Same operands for {} in {}'.format(b, source_file)
```

Those 20 lines of code found 1 bug in GNAT, 3 bugs in CodePeer, and 1 bug in GPS (despite extensive testing and static analysis).

More info on our blog

Semantic based static analyzers

TODO: fill when Romain sends

Copy paste detector

Current applications

- Inside Adacore: new versions of GNATmetric, GNATStub, GNATpp
- Outside: Clients using it in production for various needs such as:
  - Code instrumentation
  - Automatic refactorings
  - Generation of serializers/deserializers

Conclusion

- Sources are on GitHub
- Come open issues and create pull requests!