Libadalang Tutorial - Ada Europe 2018

Pierre-Marie de Rodat   Raphaël Amiard

Software Engineers at AdaCore

AdaCore

Previously in Ada Europe: Libadalang

- A library that allows users to query/alter data about Ada sources
- Both low & high level APIS:
    - What is the type of this expression?
    - How many references to this variable?
    - Give me the source location of this token
    - Rename this entity
    - Etc.

- Multi-language: Easy binding generation to other languages/ecosystems
    - Today: Python, Ada, C

- Easy scripting: Be able to create a prototype quickly & interactively
- API is still evolving! Hopefully reaching stability in October 2018

AdaCore

Today's tutorial

What we will do

- How to use Libadalang in order to create Ada tooling
- Computation of metrics based on syntax and semantics
- Perform automatic refactorings
- Focused on the Ada API

Requirements

- GNAT Community 2018 (https://www.adacore.com/download/)
- Build Libadalang yourself, or get and install:
  - On Linux: URL
  - On Windows: URL

AdaCore

API Concepts

- `Libadalang.Analysis.Analysis_Context` type
- Holder for all computations in Libadalang
- `Create` and `Destroy`
- Owns analysis units

- Libadalang.Analysis.Analysis_Unit type
- Owns for tokens, parsing tree and semantic data for a source file
- Get_From_File, Get_From_Buffer, Get_From_Provider

Nodes

- Libadalang.Analysis.Ada_Node type and derivations
- Nodes for the parsing tree, plus generic instantiation context
- Accessors common to all nodes: Kind, Parent, Children, Sloc_Range, Text, …
- Special nodes:
    - lists contain variable number of nodes
    - token nodes have no child, only a label (e.g. identifiers, string literals)

Node fields

- `Libadalang.Analysis.F_*` functions
- `F_` = field: let one go down the syntax tree
- All take a node and return another (possibly null) node
- For instance: `F_Type_Expr` or `F_Has_Aliased` for object declarations (`Object_Decl` nodes)

Node properties

- Libadalang.Analysis.P_* functions
- P_ = property: dynamic evaluation for name resolution, implemented on top of syntax fields
- For instance: P_Referenced_Decl, P_Primitive_Subp_Of, …

- Once the analysis context is created, start a rewriting session
- Do modifications (create new nodes, replace, remove) on a "virtual" tree
- Original one is unmodified, so name resolution is still available
- Once done, apply the rewriting: modifies analysis units in place

- All in Libadalang.Rewriting package
- Rewriting_Handle for the rewriting session
- Unit_Rewriting_Handle and Node_Rewriting_Handle for the virtual tree
- Handle/Unit/Node functions to go back and forth between virtual and original trees

AdaCore

Rewriting: operations

- Root/Set_Root to get/set analysis unit root node
- Child/Set_Child to get/set node children
- Set_Text to set text of token nodes
- {Insert,Append,Remove}_Child to rewrite lists
- Clone/Create_* to create new nodes
- Create_From_Template to create trees of new nodes from text

Rewriting help: introspection

- Get back and forth between field name and child index
    - For instance: `F_Suffix` is the second field of the `Call_Expr` node
- All in `Libadalang.Introspection` package
- `Index (Call_Expr_F_Suffix)` returns 2

AdaCore

```
with "libadalang";

project Exercizes is
  for Main use ("exercize01.adb");
end Exercizes;
```

```
gprbuild -Pexercizes.gpr -XLIBRARY_TYPE=relocatable -XXMLADA_BUILD=relocatable -p
```

AdaCore