Langkit

source code analyzers for the masses

Pierre-Marie de Rodat   Raphaël Amiard

Software Engineers at AdaCore

AdaCore

Langkit: A meta compiler

High level vision

A collection of DSLs to implement language parsing and analysis front-ends.

Front ends generated by Langkit could be the basis for:

[BULLET POINTS]

Original use case: Libadalang

[Description of Libadalang]

The DSL

Syntax

- Python-based DSL for now (will self-host one day!)
- Really several sub-DSLs: each has its own purpose

DSL Episode 1: Lexing

- Define a list of token kinds:

```
from langkit.lexer import LexerTokenn, WithText, WithSymbol

class MyTokens(LexerToken):
    Def = WithText()
    Identifier = WithSymbol()
    # ...
```

- Provide regexp-based scanning rules to produce them:

```
from langkit.lexer import Lexer, Literal, Pattern

my_lexer = Lexer(MyToken)
my_lexer.add_rules(
    (Literal('def'), MyTokens.Def),
    (Pattern(r'[a-zA-Z][a-zA-Z0-9]*',
     MyTokens.Identifier)),
    # ...
)
```

DSL Episode 2: Tree

- Define lists of AST nodes the parser can produce:

```
from langkit.dsl import ASTNode, Field

class RootNode(ASTNode):
    pass

class Name(RootNode):
    token_node = True

class Def(RootNode):
    name = Field()

# ...
```

- AST nodes inheritance tree
- Nodes can be abstract
- Optional type annotations on Field

DSL Episode 3: Parsing

- Recursive descent parser combinators (sequences, lists, optional parts, alternatives, …)
- Packrat parsers
- Add lists of parsing rules
- Specify one default starting one

```
from langkit.parsers import Grammar, List

g = Grammar('main_rule')
g.add_rules(main_rule=List(g.def_rule),
            name=Name(MyTokens.Identifier),
            def_rule=Def('def', g.name),
            # ...
)
```

- Compiling the grammar:
  - infers AST node types Field annotations not present if not present;
  - checks consistency otherwise.

DSL Episode 4: Scoping

- Sub-DSL inside the AST node declarations
- Foundation for semantic analysis
- Create name/AST nodes mappings: lexical environments

```
from langkit.dsl import T
from langkit.envs import EnvSpec, add_env, add_to_env
from langkit.expressions import New, Self

class Function(RootNode):
    local_vars = Field()
    env_spec = EnvSpec(add_env())

class VariableDeclaration(RootNode):
    name = Field()
    env_spec = EnvSpec(add_to_env(mappings=New(
        T.env_assoc, key=Self.name.symbol, val=Self)))
```

DSL Episode 5: Semantic analysis

- Sub-DSL inside AST node declarations
- Create kind of methods on AST nodes
- Public methods: user API for semantic analysis
- Private ones: implementation detail, hidden from users
- Functional programming language

```
from langkit.expressions import langkit_property

class VariableReference(FooNode):
    name = Field()

    @langkit_property(public=True)
    def var_decl():
        return Self.node_env.get_first(Self.name)
```

Crafted for incremental analysis

- Reloading happens a lot in IDE: performance required
- Avoid big recomputations for common operations
- No need to recompute *everything* when reloading one source file:
- Keep source file-specific data as much isolated as possible
- Reduced update process when removing/reloading source files

The generated libraries

Base library: Ada (W00T!)

Requirements for the target language:

- Fast
- Low level enough
- Memory management agnostic (no GC)
- Easy to bind to C and other languages

Candidates

- C, C++, Ada, Rust, ...

Chosen one: Ada

- Since the project is developed at AdaCore: no surprises :)

Bindings to other languages

Automatically generated C bindings

So that it is very easy to generate bindings to any languages the users wants.

First class citizen Python bindings

- Python is the de-facto scripting language of the Langkit ecosystem.
- Everything possible in Ada is possible in Python

Easy to generate bindings to new languages

- No need for external bindings generators
- Knowledge about data types, functions, memory management -> Langkit

Easy to interface

[explain ecore stuff]

Tree walking

Rewriting

Generic tools shipping with the libraries

Playground

Vim plug-ins

Language server protocol? (not done)

Demo!