Toward a Modular Approach for Type Systems and LSP generation

Federico Bruzzone
Id. Number: 27427A

Università degli Studi di Milano
Computer Science Department
MSc in Computer Science

Advisor: Prof. Walter Cazzola
Co-Advisor: Dr. Luca Favalli

15/07/2024

LM-18 - Computer science
Academic Year 2023-2024

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

Problem Statement

Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

This makes the maintainability, extensibility and reusability of the implementation difficult.

Language Server Protocol

LSP In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP).

Language Server Protocol

LSP In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP).

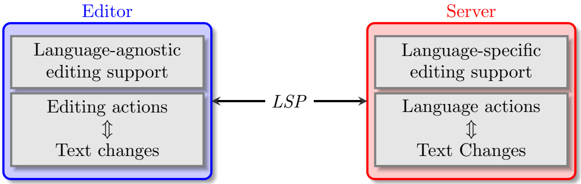
The LSP allows the communication between a Language Server and an IDE.

Language Server Protocol

LSP In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP).

The LSP allows the communication between a Language Server and an IDE.



Editor

Language-agnostic editing support

Editing actions ⇳ Text changes

LSP

Server

Language-specific editing support

Language actions ⇳ Text Changes

Intrinsic properties:

- Language-agnostic
- IDE-agnostic
- Asynchronous
- Text-based

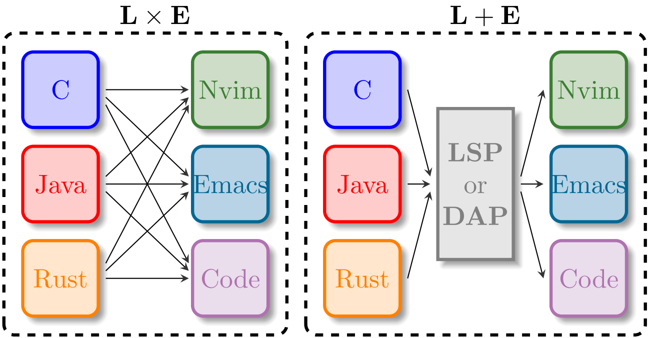
Features:

- Diagnostics
- Hover
- Go to definition
- Find references

Initially implemented for Visual Studio Code, the LSP has been adopted by several IDEs and programming languages.

Problem Statement
LSP
In a Nutshell
The Reductions of Combinations
An Achievement
FOP
LWs
Scientific Contribution
Type System Components
Checking and Inference
Modularization
LSP in Action
Conclusions

Initially implemented for Visual Studio Code, the LSP has been adopted by several IDEs and programming languages.



Language Server Protocol

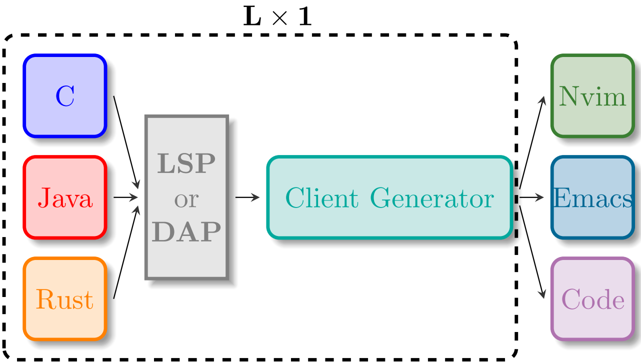
What would be an important achievement?

Reducing the number of combinations between Language Servers and IDEs.

Language Server Protocol

What would be an important achievement?

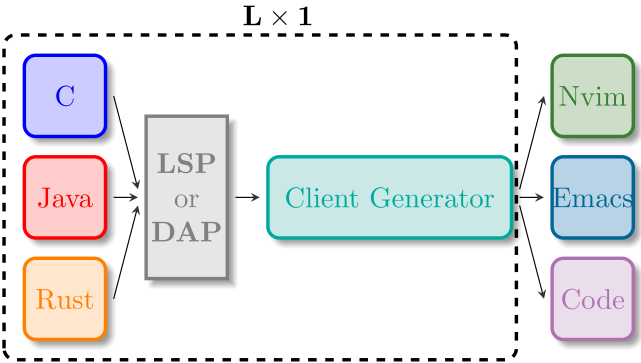
Reducing the number of combinations between Language Servers and IDEs.



Language Server Protocol

What would be an important achievement?

Reducing the number of combinations between Language Servers and IDEs.



Spoiler: It is possible! and we have done better than that.

Feature-Oriented Programming

Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

Feature-Oriented Programming

Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

- Feature is a unit of functionality that satisfies a requirement.
- Feature Model is a model that represents the variability of the SPL.
- Feature Configuration is a set of features that compose a product.

Language Workbenches

Language Workbenches (LWs) are tools that allow the development of programming languages, both GPLs and DSLs.

Language Workbench	Modularization Supp.	Precompiled Feature Supp.	Native IDE Gen.	LSP Gen.	LSP Mod.
JustAdd	◐	○	○	○	○
Melange	Ⓐ	○	3rd p.	☆	☆
MontiCore	◐	◐	●	○	○
MPS	Ⓐ	○	●	☆	☆
Rascal	○	○	●	○	○
Spoofax	Ⓐ	◐	●	☆	☆
Xtext	○	◐	●	●	○
Neverlang	Ⓥ	●	○	✭	✭

● Full support

○ No support

◐ Limited support

Ⓥ Fine-grained mod.

Ⓐ Coarse-grained mod.

✭ My contribution

☆ Future Work

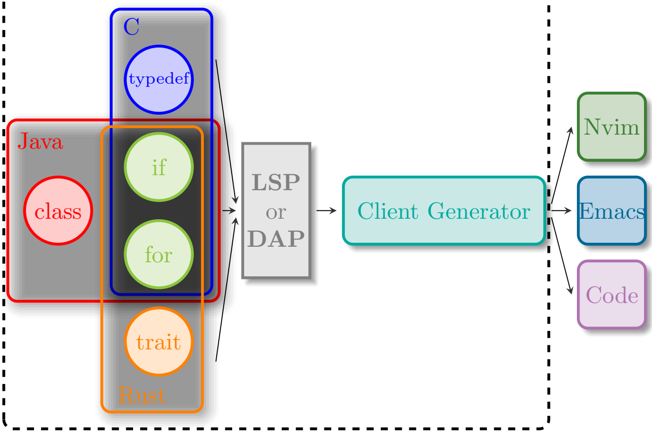
3rd p. Third-party

Scientific Contribution

- Methodology for whole LWs that support at least component modularization.
- Type System and LSP Modularization.
- DSL (about 2k LoC) for Type System definition.
- LSP generation for Neverlang languages.
- Client and Syntax Highlighting generation reducing the number of combinations.
- Implementation of a Java Library (about 6k LoC) for Neverlang to support the type system for every language developed with Neverlang.
- 3 use cases to show the effectiveness of the methodology.

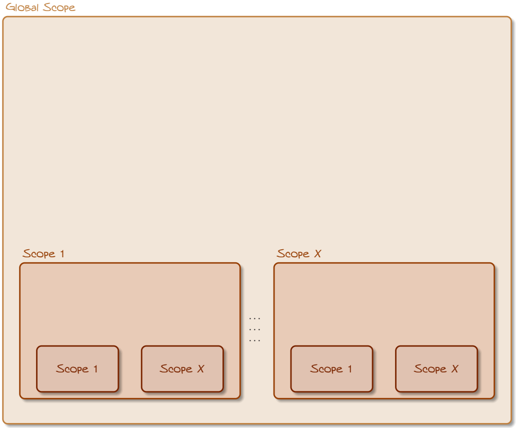
Scientific Contribution

$\mathbf{N} \times \mathbf{1}$ where $\mathbf{N} << \mathbf{L}$



Scientific Contribution

Type System Components



Global Scope

Scope 1

Scope 1

Scope *X*

. . .

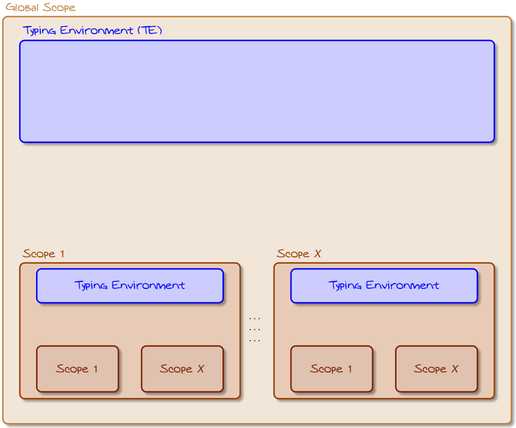
Scope *X*

Scope 1

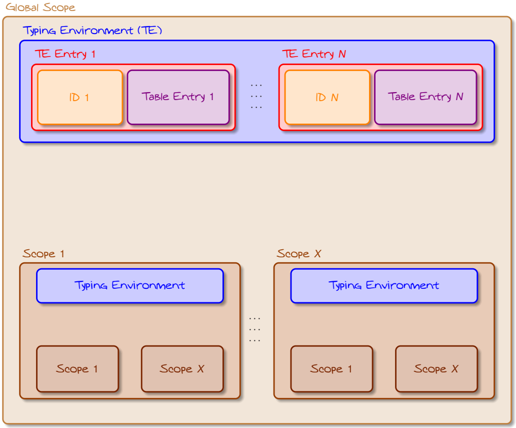
Scope *X*

Scientific Contribution

Type System Components



Scientific Contribution
Type System Components



Global Scope

Typing Environment (TE)

TE Entry 1

ID 1

Table Entry 1

...

TE Entry *N*

ID *N*

Table Entry *N*

Scope 1

Typing Environment

Scope 1

Scope *X*

...

Scope *X*

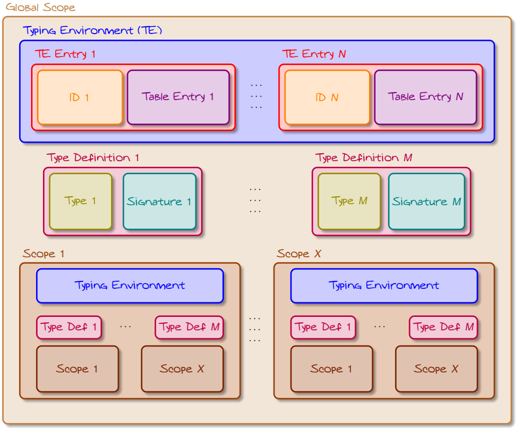
Typing Environment

Scope 1

Scope *X*

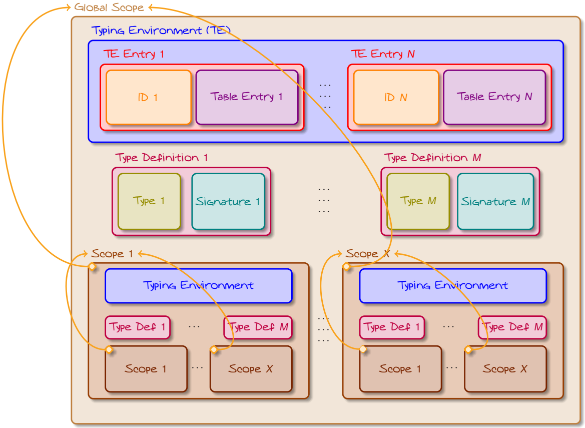
Scientific Contribution

Type System Components



Scientific Contribution

Type System Components



Global Scope

Typing Environment (TE)

TE Entry 1

ID 1

Table Entry 1

TE Entry *N*

ID *N*

Table Entry *N*

Type Definition 1

Type 1

Signature 1

Type Definition *M*

Type *M*

Signature *M*

Scope 1

Typing Environment

Type Def 1 ⋯ Type Def *M*

Scope 1 ⋯ Scope *X*

Scope *X*

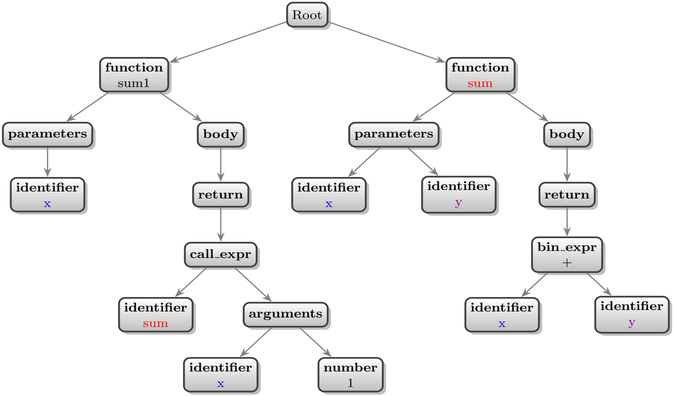
Typing Environment

Type Def 1 ⋯ Type Def *M*

Scope 1 ⋯ Scope *X*

Scientific Contribution
Type Checking and Type Inference

```
function sum1(x) {
    return sum(x, 1);
}

function sum(x, y) {
    return x + y;
}
```

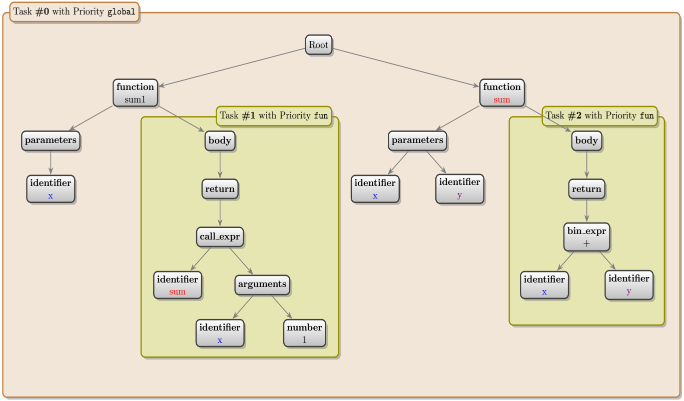


Scientific Contribution

Type Checking and Type Inference

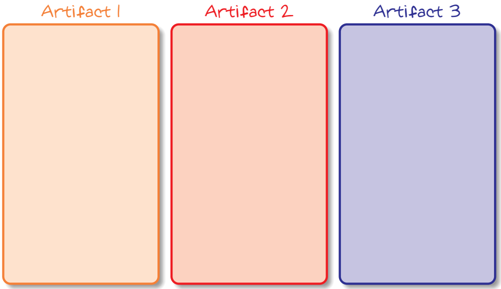
```
function sum1(x) {
    return sum(x, 1);
}

function sum(x, y) {
    return x + y;
}
```

- Compilation Unit
- Compilation Unit Task
- Compilation Helper



Scientific Contribution

TSs and LSP Modularization

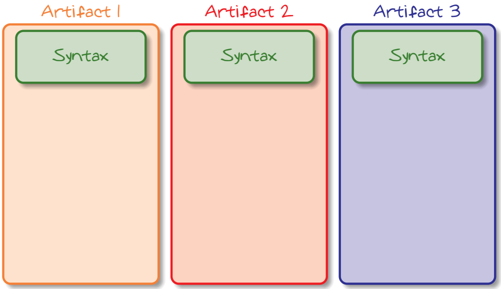


Artifact 1

Artifact 2

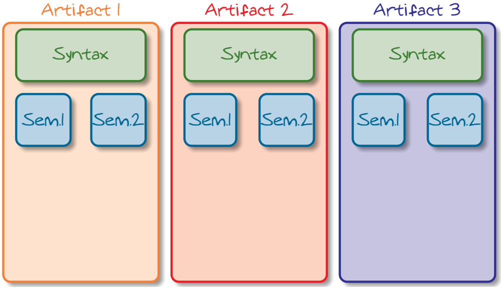
Artifact 3

Scientific Contribution
TSs and LSP Modularization



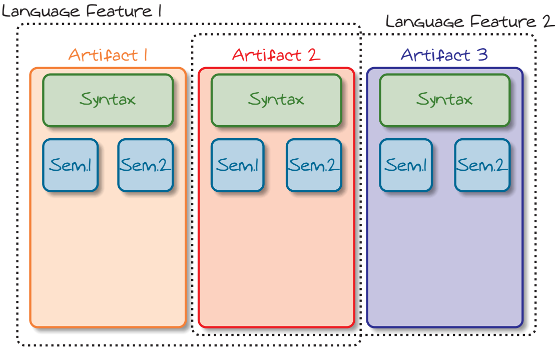
Scientific Contribution

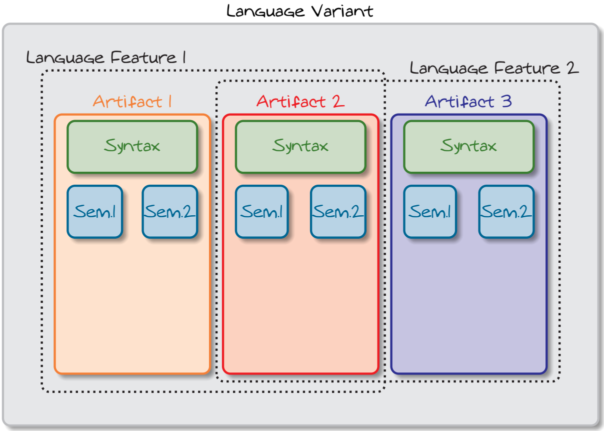
TSs and LSP Modularization



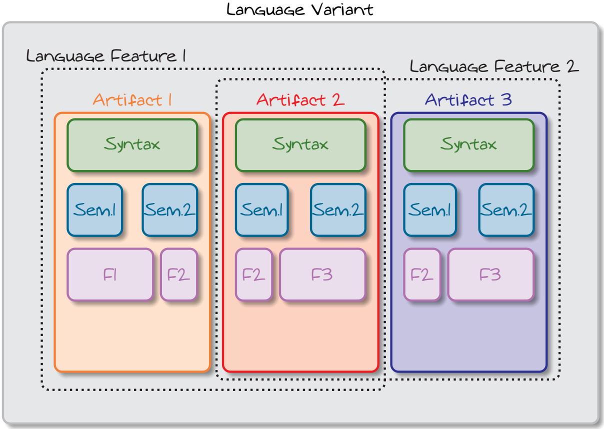
Scientific Contribution

TSs and LSP Modularization

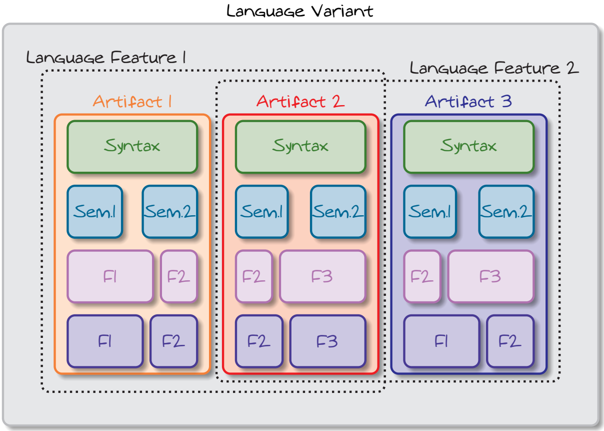



Language Variant
Language Feature 1
Language Feature 2
Artifact 1
Artifact 2
Artifact 3
Syntax
Sem.1
Sem.2
Syntax
Sem.1
Sem.2
Syntax
Sem.1
Sem.2

Scientific Contribution
TSs and LSP Modularization

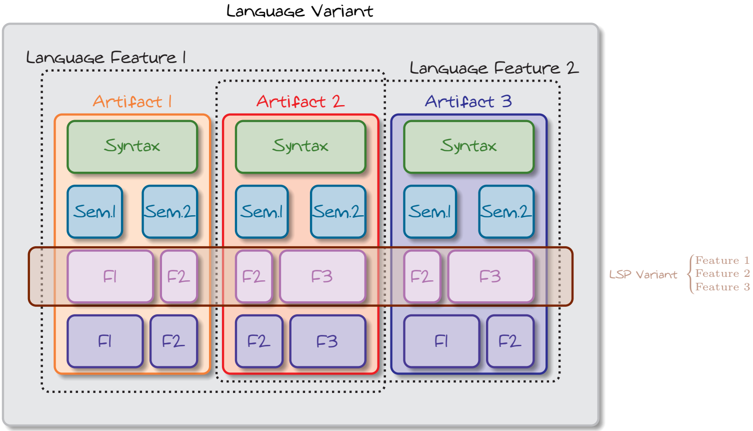


Scientific Contribution
TSs and LSP Modularization



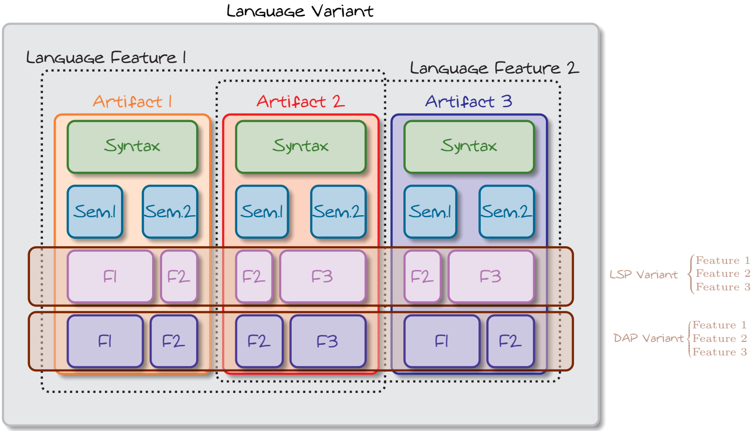
Scientific Contribution

TSs and LSP Modularization



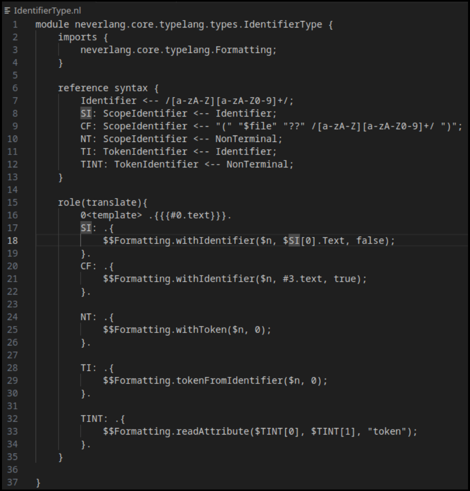
Scientific Contribution

TSs and LSP Modularization



LSP in Action

VSCode client

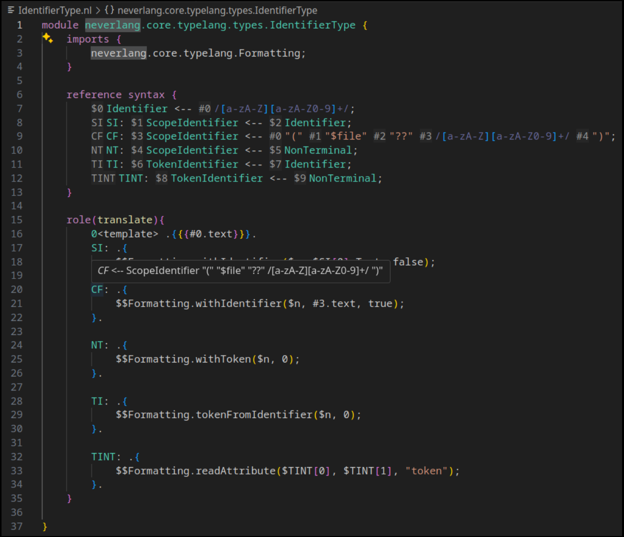


IdentifierType.nl

```
module neverlang.core.typelang.types.IdentifierType {
    imports {
        neverlang.core.typelang.Formatting;
    }

    reference syntax {
        Identifier <-- /[a-zA-Z][a-zA-Z0-9]+/;
        SI: ScopeIdentifier <-- Identifier;
        CF: ScopeIdentifier <-- "(" "$file" "??" /[a-zA-Z][a-zA-Z0-9]+/ ")";
        NT: ScopeIdentifier <-- NonTerminal;
        TI: TokenIdentifier <-- Identifier;
        TINT: TokenIdentifier <-- NonTerminal;
    }

    role(translate){
        0<template> .{{{#0.text}}}.
        SI: .{
            $$Formatting.withIdentifier($n, $SI[0].Text, false);
        }.
        CF: .{
            $$Formatting.withIdentifier($n, #3.text, true);
        }.

        NT: .{
            $$Formatting.withToken($n, 0);
        }.

        TI: .{
            $$Formatting.tokenFromIdentifier($n, 0);
        }.

        TINT: .{
            $$Formatting.readAttribute($TINT[0], $TINT[1], "token");
        }.
    }

}
```

LSP in Action

VSCode client



IdentifierType.nl > {} neverlang.core.typelang.types.IdentifierType

```
module neverlang.core.typelang.types.IdentifierType {
    imports {
        neverlang.core.typelang.Formatting;
    }

    reference syntax {
        $0 Identifier <-- #0 /[a-zA-Z][a-zA-Z0-9]+/;
        SI SI: $1 ScopeIdentifier <-- $2 Identifier;
        CF CF: $3 ScopeIdentifier <-- #0 "(" #1 "$file" #2 "??" #3 /[a-zA-Z][a-zA-Z0-9]+/ #4 ")";
        NT NT: $4 ScopeIdentifier <-- $5 NonTerminal;
        TI TI: $6 TokenIdentifier <-- $7 Identifier;
        TINT TINT: $8 TokenIdentifier <-- $9 NonTerminal;
    }

    role(translate){
        0<template> .{{{#0.text}}}.
        SI: .{
            $$Formatting.withIdentifier($n, $SI[0].Text, false);

        CF: .{
            $$Formatting.withIdentifier($n, #3.text, true);
        }.

        NT: .{
            $$Formatting.withToken($n, 0);
        }.

        TI: .{
            $$Formatting.tokenFromIdentifier($n, 0);
        }.

        TINT: .{
            $$Formatting.readAttribute($TINT[0], $TINT[1], "token");
        }.
    }

}
```

CF <-- ScopeIdentifier "(" "$file" "??" /[a-zA-Z][a-zA-Z0-9]+/ ")"

LSP in Action

Neovim client

Demonstration

Conclusions

Interesting results:

- We are writing an article (Code Less to Code More) to be submitted to JSS.

Interesting twist:

- Recycling the code of the TS to define a new compilation phase inside of Neverlang.

Future work:

- Define the same methodology for the DAP.

Thanks for your attention!

Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

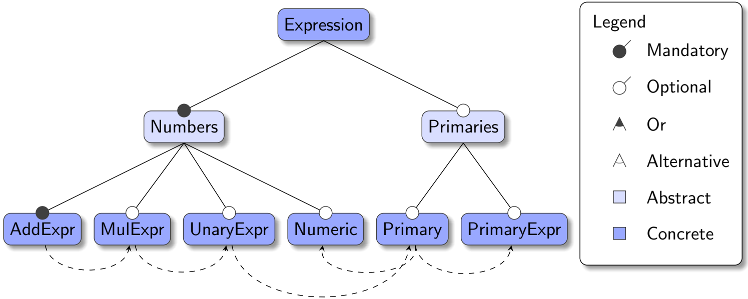
Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

- SPLs defines a family of software products.
- SPLs is described by a Feature Model.
- A Feature Model describes the variability of the software.
- SPL variants are generated by selecting a set of features.
- A feature (or artifact) is a first-class entity in SPLs.

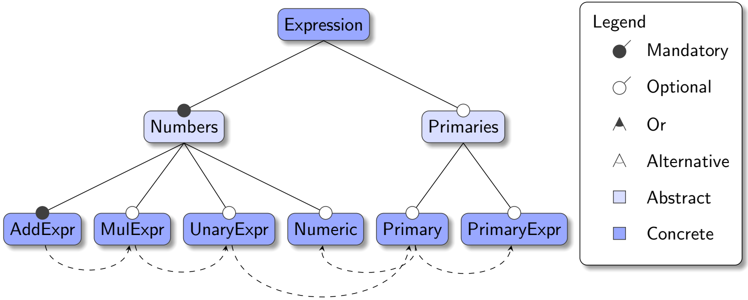
Language Product Lines

Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Language Product Lines

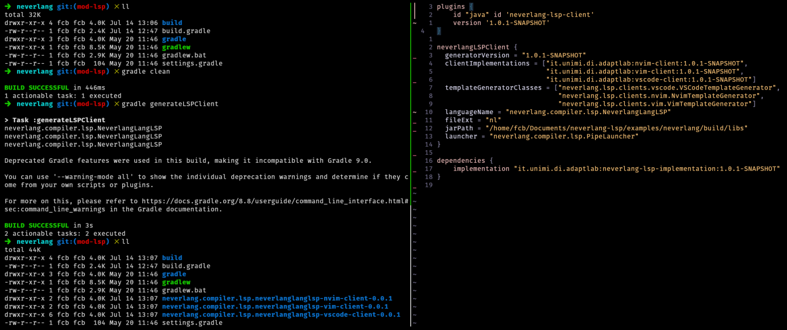
Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Some achievements:

- Bottom-up approach to language implementation
- Reusability of language artifacts
- Multiple variants of the same language
- Language Workbenches come to the rescue

Clients generation



```
➜ neverlang git:(mod-lsp) ✗ ll
total 32K
drwxr-xr-x 4 fcb fcb 4.0K Jul 14 13:06 build
-rw-r--r-- 1 fcb fcb 2.4K Jul 14 12:47 build.gradle
drwxr-xr-x 3 fcb fcb 4.0K May 20 11:46 gradle
-rwxr-xr-x 1 fcb fcb 8.5K May 20 11:46 gradlew
-rw-r--r-- 1 fcb fcb 2.9K May 20 11:46 gradlew.bat
-rw-r--r-- 1 fcb fcb  104 May 20 11:46 settings.gradle
➜ neverlang git:(mod-lsp) ✗ gradle clean

BUILD SUCCESSFUL in 446ms
1 actionable task: 1 executed
➜ neverlang git:(mod-lsp) ✗ gradle generateLSPClient

> Task :generateLSPClient
neverlang.compiler.lsp.NeverlangLangLSP
neverlang.compiler.lsp.NeverlangLangLSP
neverlang.compiler.lsp.NeverlangLangLSP

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they c
ome from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.8/userguide/command_line_interface.html#
sec:command_line_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 3s
2 actionable tasks: 2 executed
➜ neverlang git:(mod-lsp) ✗ ll
total 44K
drwxr-xr-x 4 fcb fcb 4.0K Jul 14 13:07 build
-rw-r--r-- 1 fcb fcb 2.4K Jul 14 12:47 build.gradle
drwxr-xr-x 3 fcb fcb 4.0K May 20 11:46 gradle
-rwxr-xr-x 1 fcb fcb 8.5K May 20 11:46 gradlew
-rw-r--r-- 1 fcb fcb 2.9K May 20 11:46 gradlew.bat
drwxr-xr-x 2 fcb fcb 4.0K Jul 14 13:07 neverlang.compiler.lsp.neverlanglanglsp-nvim-client-0.0.1
drwxr-xr-x 2 fcb fcb 4.0K Jul 14 13:07 neverlang.compiler.lsp.neverlanglanglsp-vim-client-0.0.1
drwxr-xr-x 6 fcb fcb 4.0K Jul 14 13:07 neverlang.compiler.lsp.neverlanglanglsp-vscode-client-0.0.1
-rw-r--r-- 1 fcb fcb  104 May 20 11:46 settings.gradle
```

```
plugins {
    id "java" id 'neverlang-lsp-client'
    version '1.0.1-SNAPSHOT'
}

neverlangLSPClient {
  generatorVersion = "1.0.1-SNAPSHOT"
  clientImplementations = ["it.unimi.di.adaptlab:nvim-client:1.0.1-SNAPSHOT",
                           "it.unimi.di.adaptlab:vim-client:1.0.1-SNAPSHOT",
                           "it.unimi.di.adaptlab:vscode-client:1.0.1-SNAPSHOT"]
  templateGeneratorClasses = ["neverlang.lsp.clients.vscode.VSCodeTemplateGenerator",
                              "neverlang.lsp.clients.nvim.NvimTemplateGenerator",
                              "neverlang.lsp.clients.vim.VimTemplateGenerator"]
  languageName = "neverlang.compiler.lsp.NeverlangLangLSP"
  fileExt = "nl"
  jarPath = "/home/fcb/Documents/neverlang-lsp/examples/neverlang/build/libs"
  launcher = "neverlang.compiler.lsp.PipeLauncher"
}

dependencies {
    implementation "it.unimi.di.adaptlab:neverlang-lsp-implementation:1.0.1-SNAPSHOT"
}
```