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 and recovery
- 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 and recovery
- IDE support
- Documentation

It is usually done in a monolithic way, 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 and recovery
- IDE support
- Documentation

It is usually done in a monolithic way, where all the aspects are tightly coupled.

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

Problem Statement
Type Systems and IDEs Support

Often some parts of compilation, such as code generation, makes use of feature-oriented programming to support different architectures.

Problem Statement
Type Systems and IDEs Support

Often some parts of compilation, such as code generation, makes use of feature-oriented programming to support different architectures.

However, the type system and the IDE support are usually implemented using a top-down approach.

Software Product Lines

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

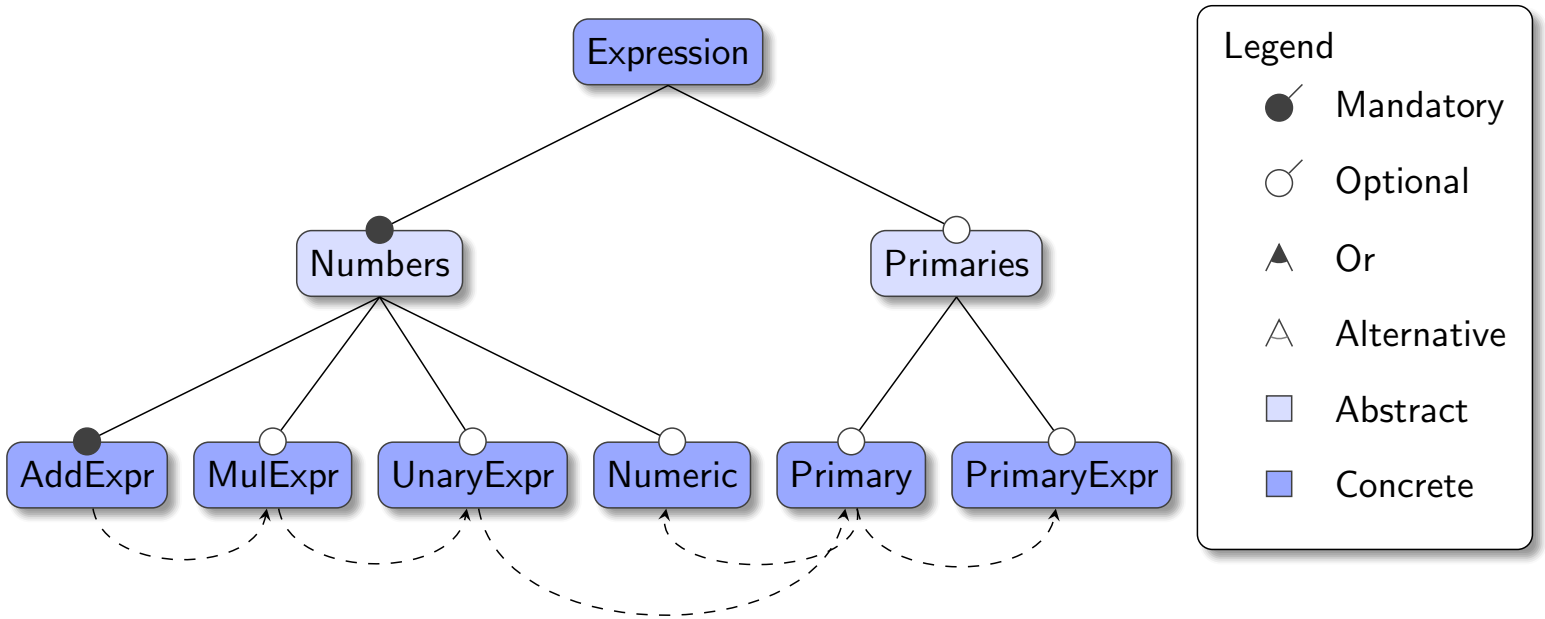
Software Product Lines

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

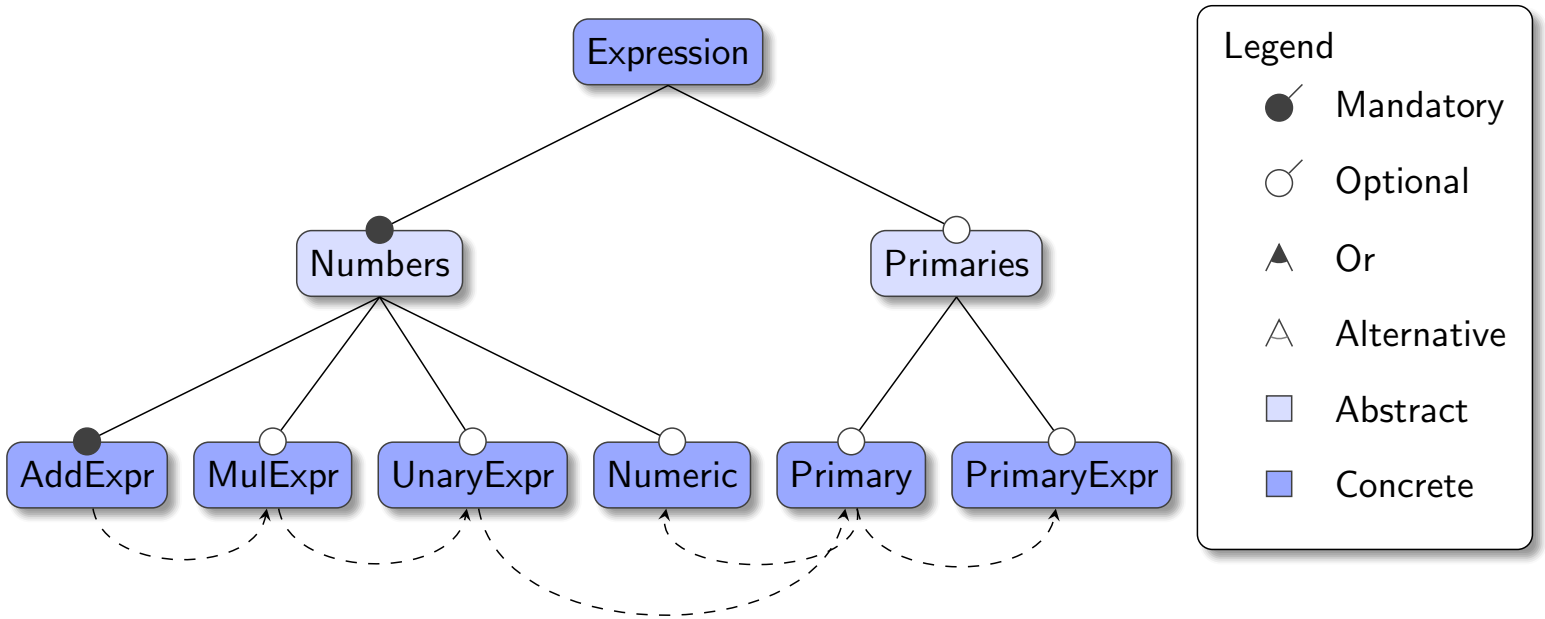
- SPLs defines a family of software products.
- SPLs is described by a Feature Model.
- A Feature Model describes the variability of the software.
- SPL variants are generated by selecting a set of features.
- A feature (or artifact) is a first-class entity in SPLs.

Software Product Lines
Language Product Lines

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



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



Some achievments:

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

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

Language Workbench	Modularization Supp.	Precompiled Feature Supp.	Native IDE gen.	LSP Gen.	LSP Mod.
JustAdd	◐	○	○	○	○
Melange	⍙	○	2rd party (EMF)	☆	☆
MontiCore	◐	◐	●	○	○
MPS	⍙	○	●	☆	☆
Rascal	○	○	●	○	○
Spoofax	⍙	◐	●	☆	☆
Xtext	○	◐	●	●	○
Neverlang	⍛	●	○	★	★

- ●Full support
- ○No support
- ◐Limited support
- ⍛ Fine-grained mod.
- ⍙ Coarse-grained mod.
- ★My contribution
- ☆ My contribution extended

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

Language Workbench	Modularization Supp.	Precompiled Feature Supp.	Native IDE gen.	LSP Gen.	LSP Mod.
JustAdd	◐	○	○	○	○
Melange	⍙	○	2rd party (EMF)	☆	☆
MontiCore	◐	◐	●	○	○
MPS	⍙	○	●	☆	☆
Rascal	○	○	●	○	○
Spoofax	⍙	◐	●	☆	☆
Xtext	○	◐	●	●	○
Neverlang	⍛	●	○	★	★

- ●Full support
- ○No support
- ◐Limited support
- ⍛ Fine-grained mod.
- ⍙ Coarse-grained mod.
- ★My contribution
- ☆ My contribution extended

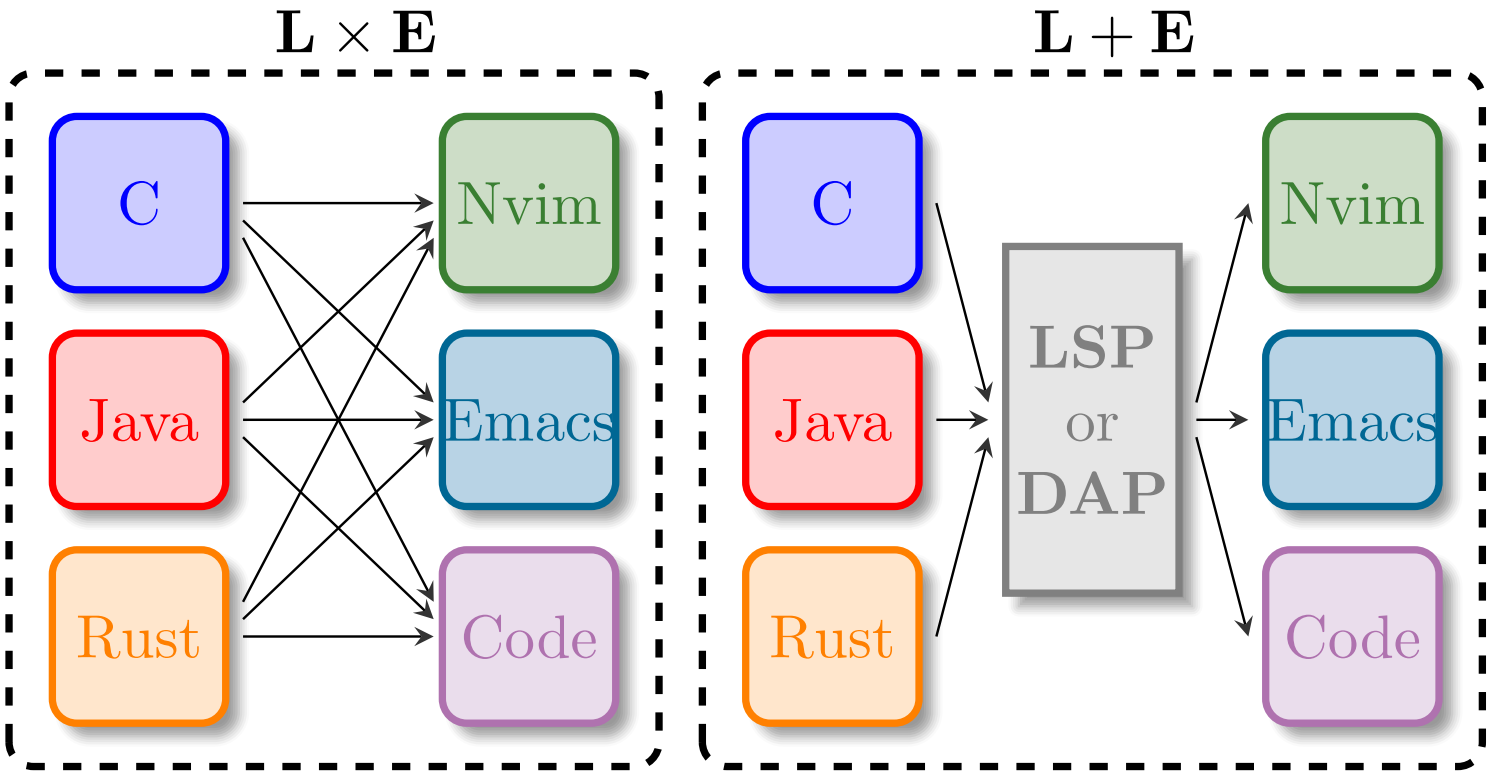
Neverlang is a language workbench, developed by the ADAPT lab, that supports the development of LPLs.

Language Server Protocol
The Reduction of Combinations

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

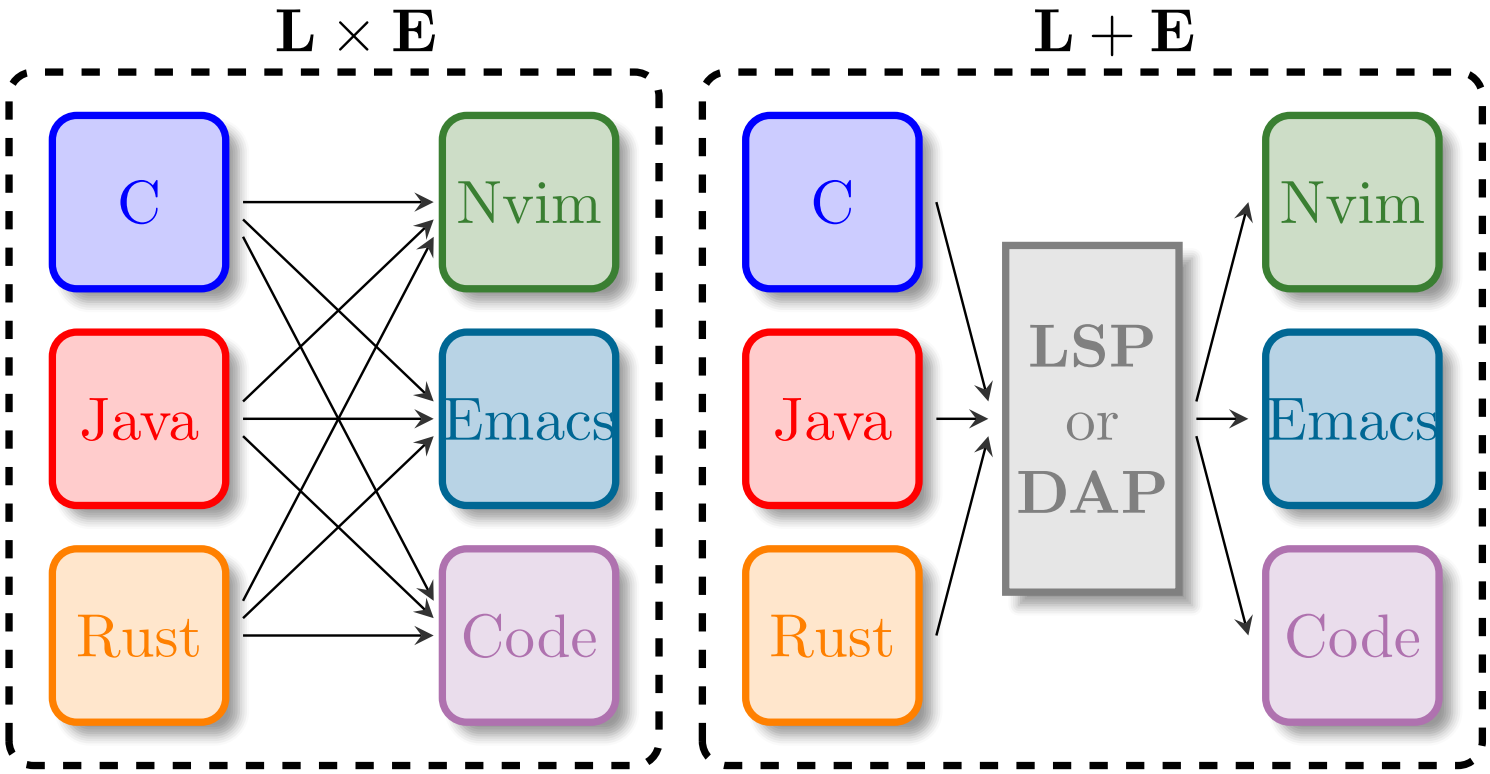
Language Server Protocol
The Reduction of Combinations

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



Language Server Protocol
The Reduction of Combinations

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



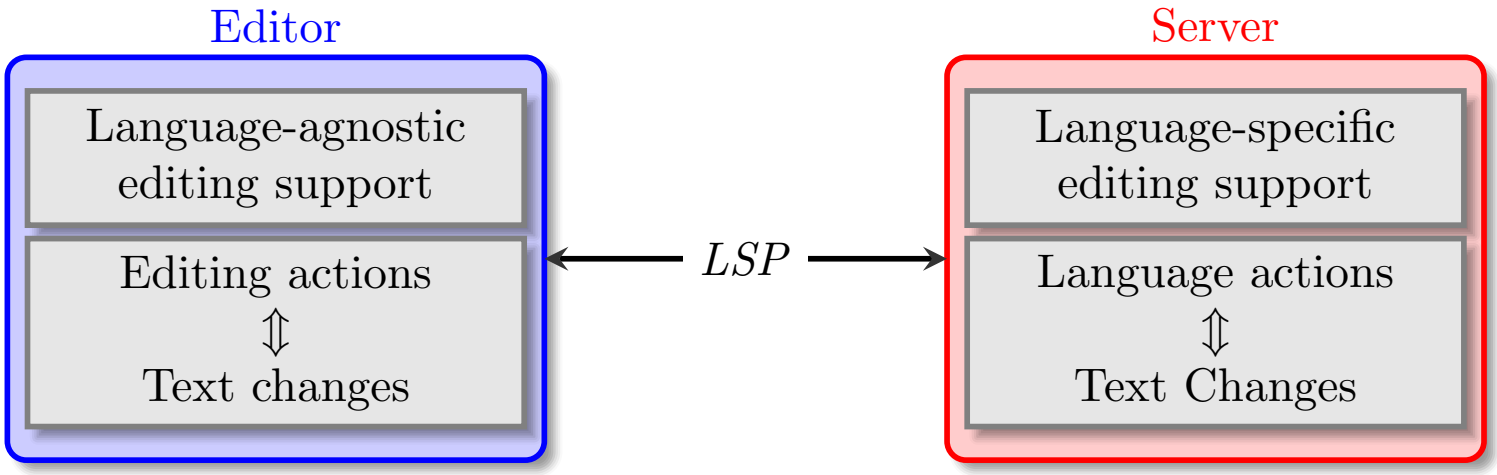
Spoiler: We have reduced the number of combinations from $\mathbf{L} \times \mathbf{E}$ to $\mathbf{N} \times \mathbf{1}$ where $\mathbf{N} \ll \mathbf{L}$.

Language Server Protocol
LSP In a Nutshell

The Language Server Protocol (LSP) is a protocol that allows the communication between a Language Server and an IDE.

Language Server Protocol
LSP In a Nutshell

The Language Server Protocol (LSP) is a protocol that allows the communication between a Language Server and an IDE.



Intrinsic properties:

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

Features:

- Diagnostics
- Hover
- Go to definition
- Find references
- Inlay hints

- Type System implementation and a Java Library for Neverlang in order to support the type system for every langauge developed with Neverlang. - Type System Modularization - LSP generation for Neverlang languages - DSL for Type System definition - Client and Syntax Highlighting generation