Incremental Variability Models for Language Composition Inference

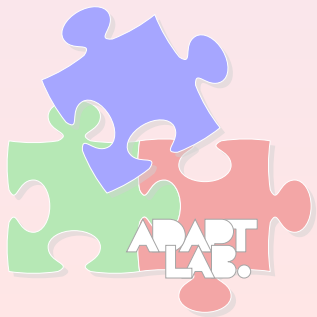
Luca Favalli

Università degli Studi di Milano
Computer Science Department

@T-LADIES kick-off meeting

Pisa, July 7th 2022

Joint work with Walter Cazzola



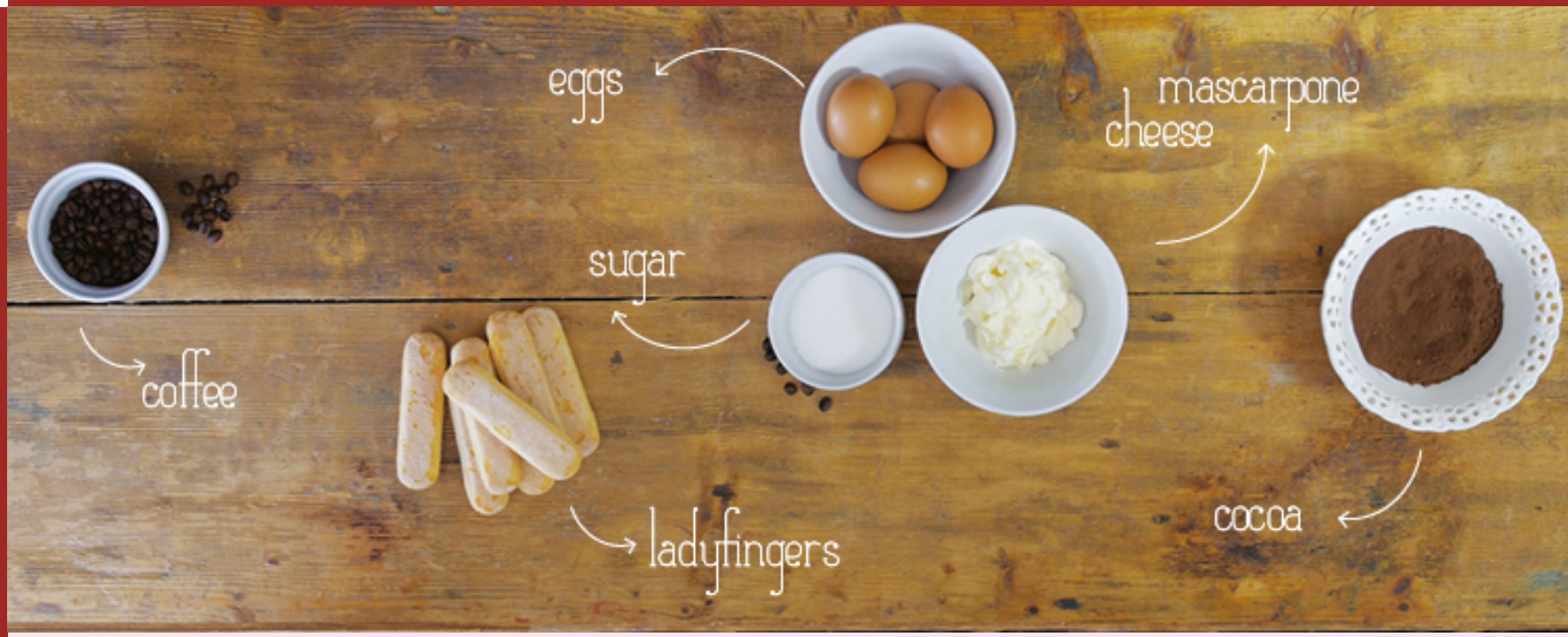
Language Product Lines
Language Families

Multiple variants of GPLs and ever more DSLs.

Examples of language families:

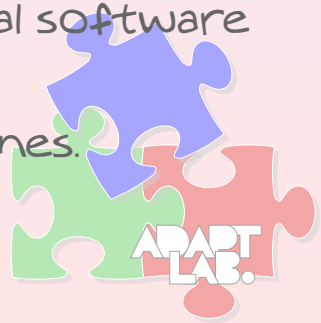
- OCL variants [Wende et al. 2009]
- Javascript subsets for teaching [Cazzola and Olivares 2016]
- VML* for variability management [Zschaler et al. 2009]
- Role-Based Programming Languages [Kühn and Cazzola 2016]

Language Product Lines

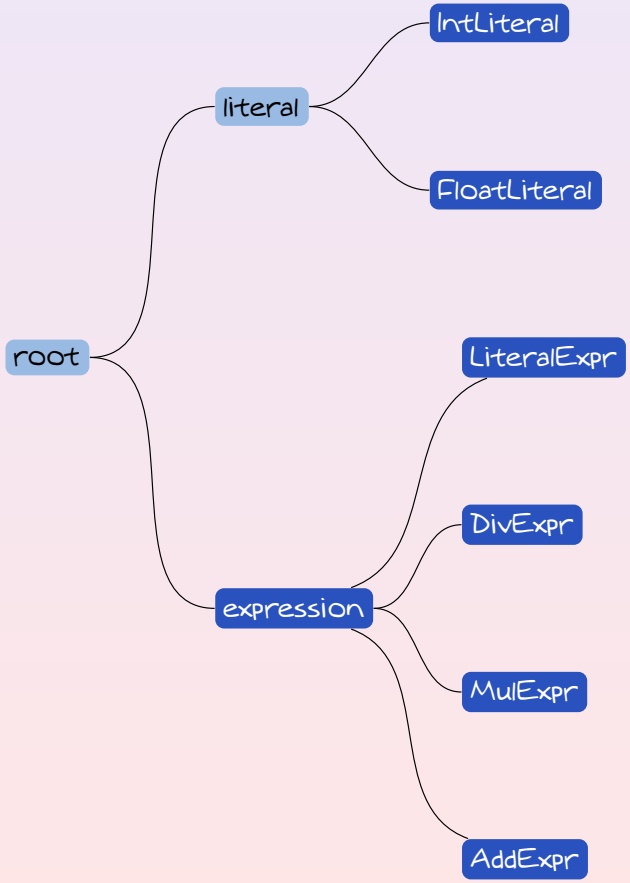


Language product lines (LPLs)

- Create language variants by choosing and picking language features.
- Resulting compilers/interpreters are products of a special software product line composing language components.
- Language variants are products of a language product lines.
- Multiple language development tools support LPLs
  - Melange, Monticore, Neverlang

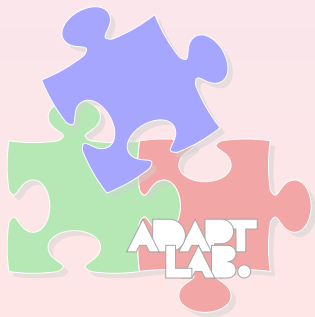


Language Families are often modeled in terms of their features using a variability model, according to formalisms such as the feature model



Example:

- (part of) the expression languages family



Language Product Lines
Problem

Benefits

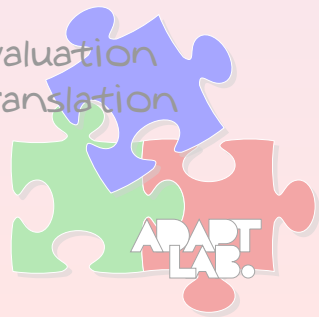
- Modular language design
- Opportunistic reuse
- Extensibility
- Implementation-agnostic language deployment
- Language-oriented programming

Limitations

- Language composition is complex
- Huge configuration space
- Dependency management
- Feature reuse outside the originally intended application

...But must importantly languages can express variability along three dimensions!

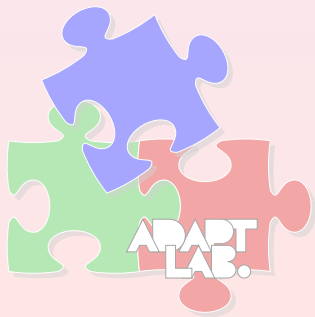
- Abstract syntax
  - constructs
  - abstractions
- Concrete syntax
  - textual
  - graphical
- Semantics
  - evaluation
  - translation



Language Product Lines
Problem

Describing all three dimensions upfront in a top-down approach is not feasible.

- Steep barrier to entry
- Error-prone
- Alignment between model and implementation
- Feature traceability
- Extending the implementations requires rewriting the model first

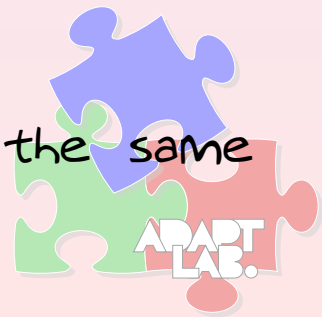


Bottom-up Language Product Line Engineering
Overview

Develop the variability model BOTTOM-UP and incrementally.

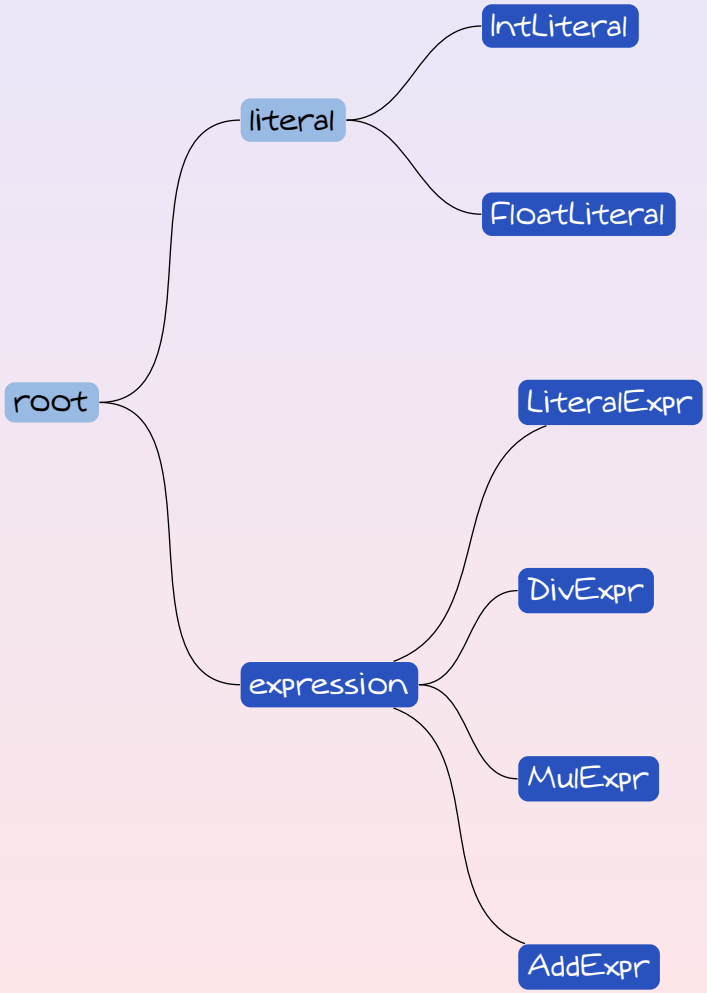
1. Implement the abstract syntax, concrete syntax and semantics of the language family
2. Extract the abstract syntax variability model from the abstract syntax implementation
3. Configure the concrete syntax using the variability model
4. Extract the language grammar
5. Configure the grammar with its semantics while solving any semantic dependencies using pre-conditions and post-conditions
6. Compose abstract syntax, concrete syntax and semantics into a language feature
7. Increment the variability model with the newly created language features

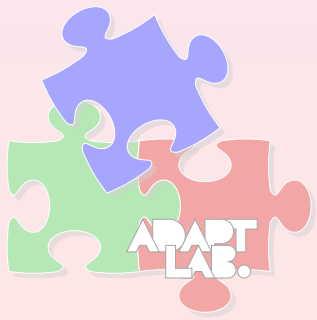
Configuration and modeling activities coexist in the same process.



Bottom-up Language Product Line Engineering

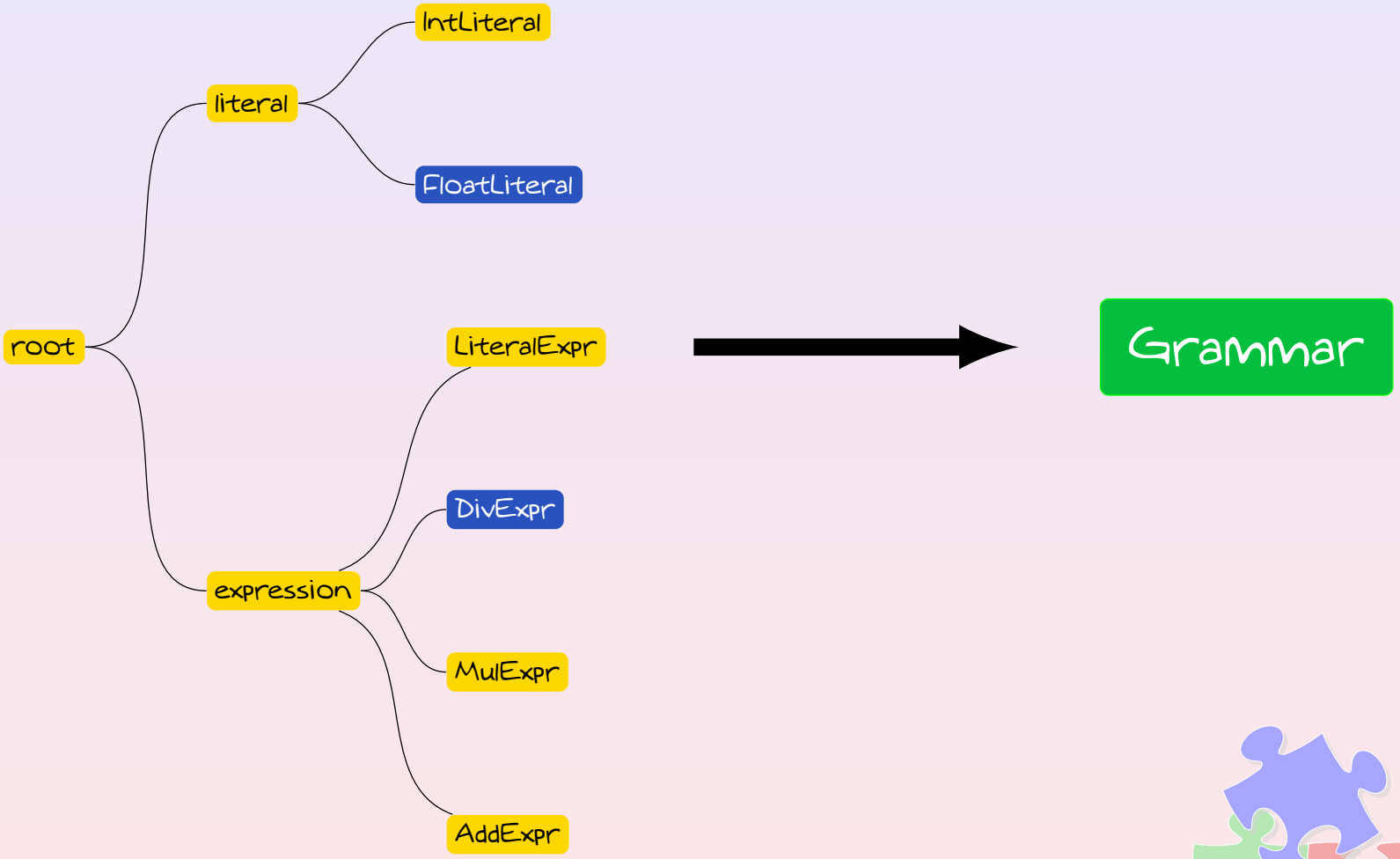
Extract the variability model from the abstract syntax

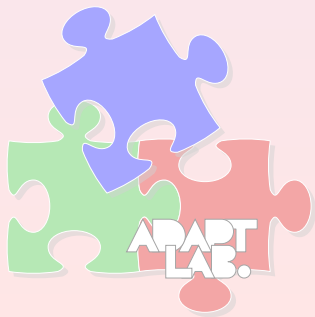




Bottom-up Language Product Line Engineering

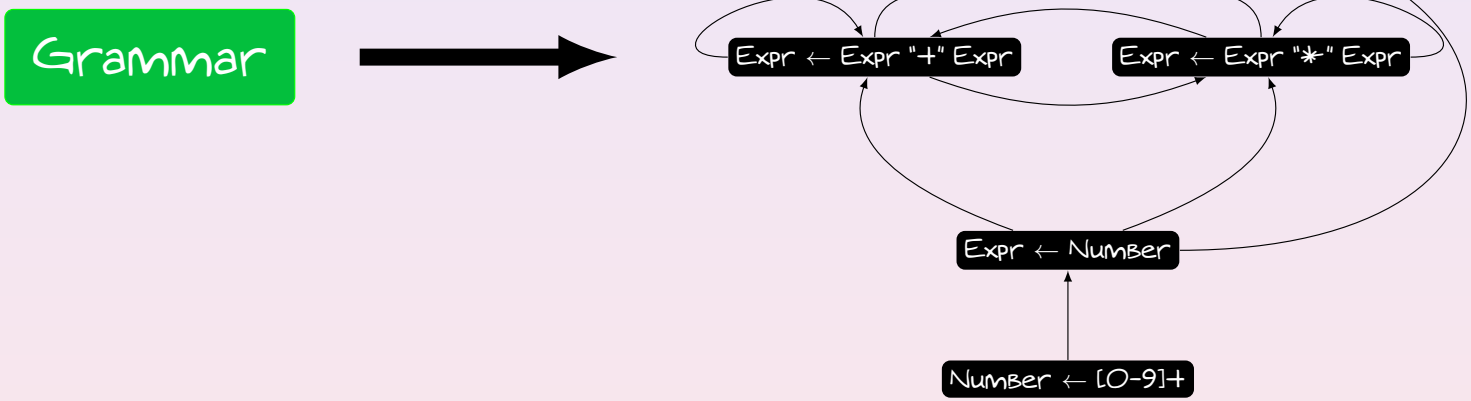
Configure the concrete syntax and extract the grammar

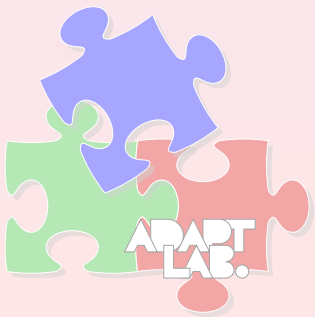




Bottom-up Language Product Line Engineering

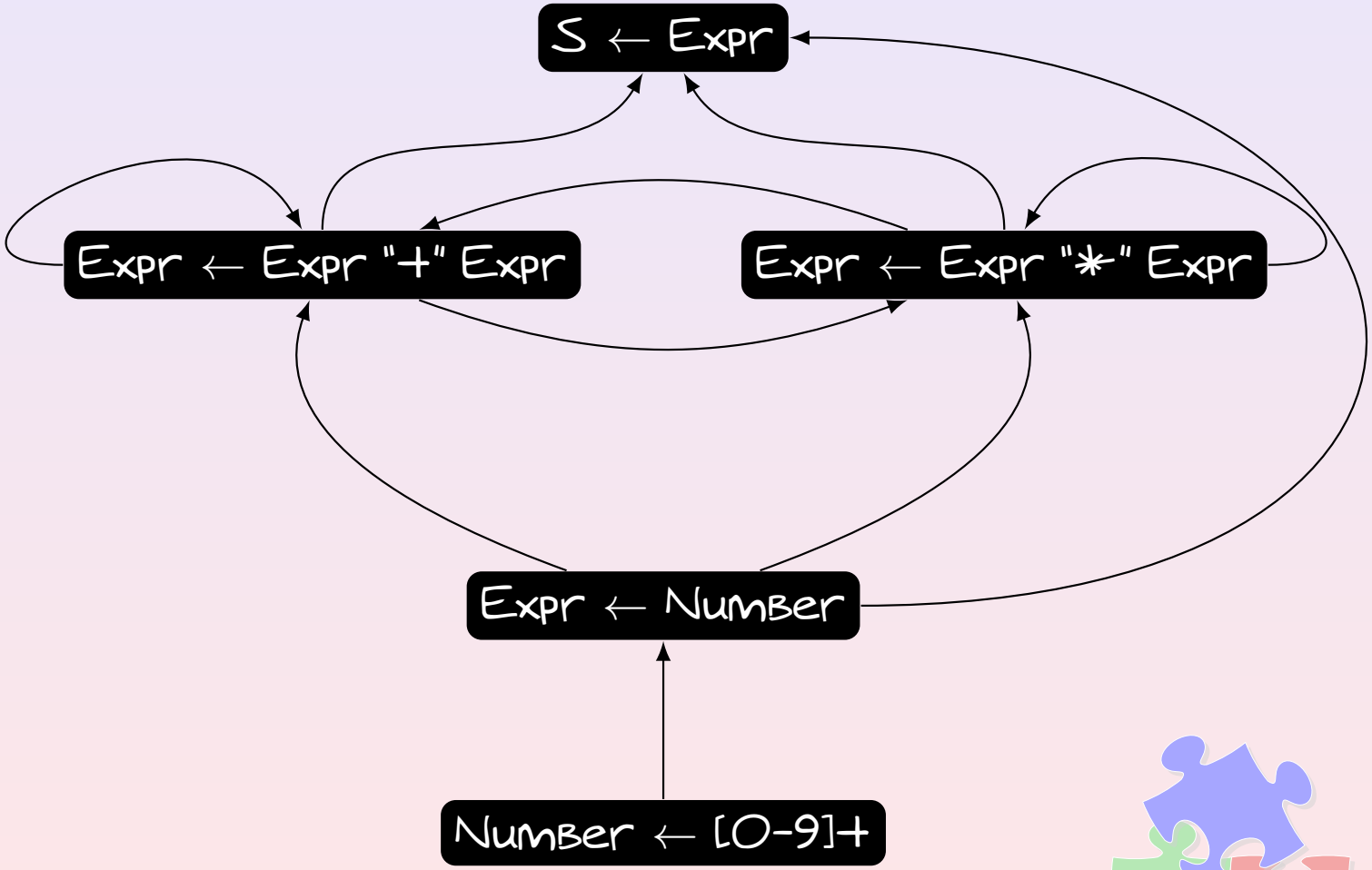
Configure the concrete syntax and extract the grammar

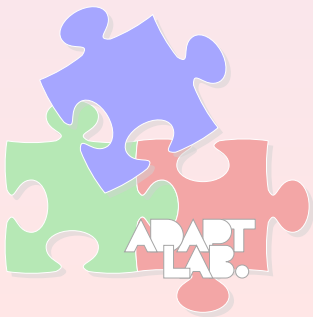




Bottom-up Language Product Line Engineering

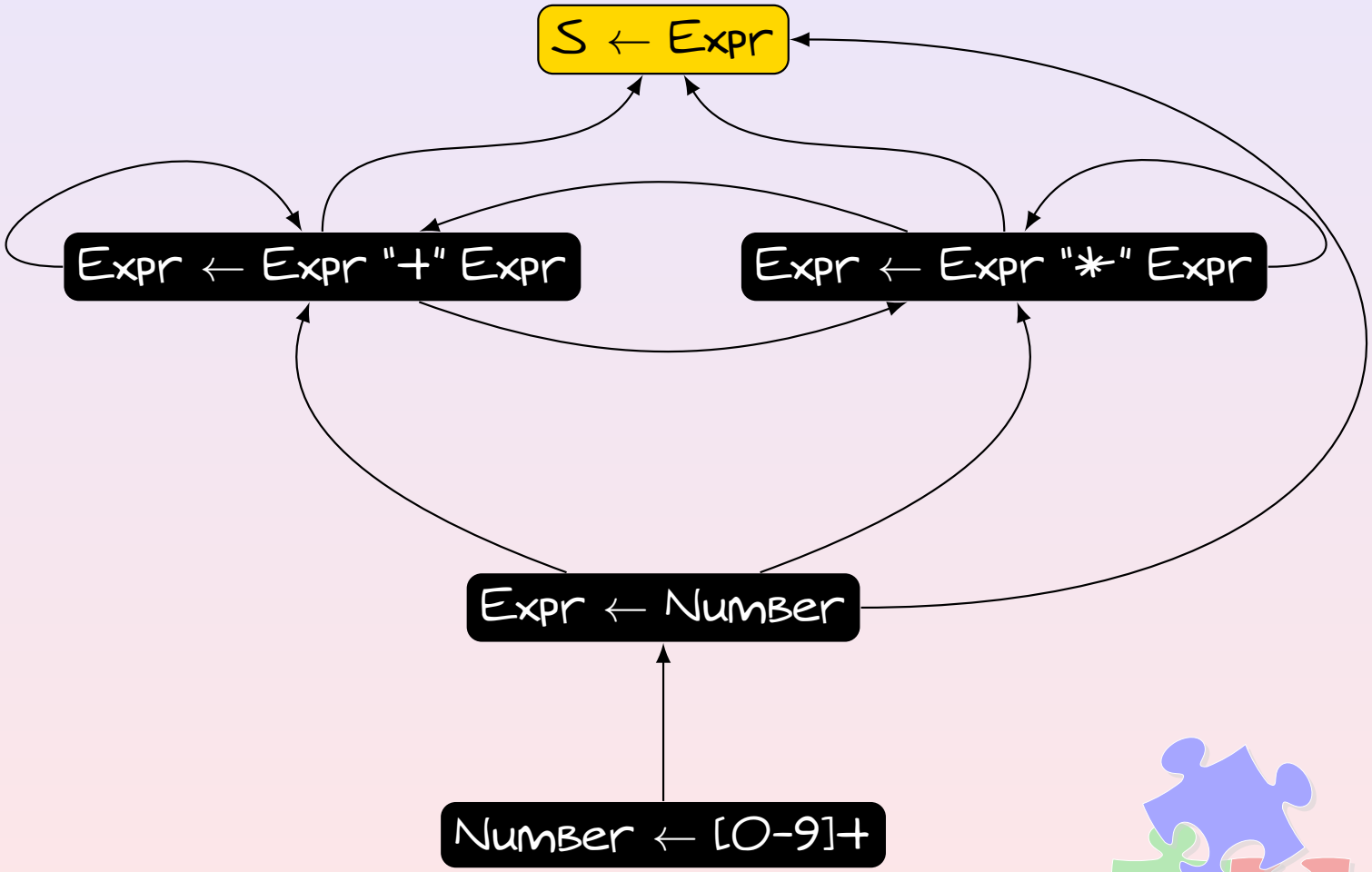
Configure the language semantics

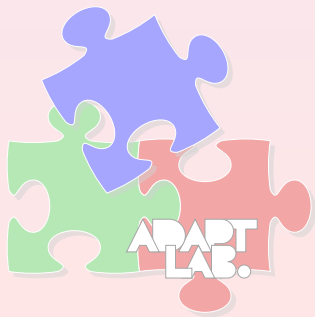




Bottom-up Language Product Line Engineering

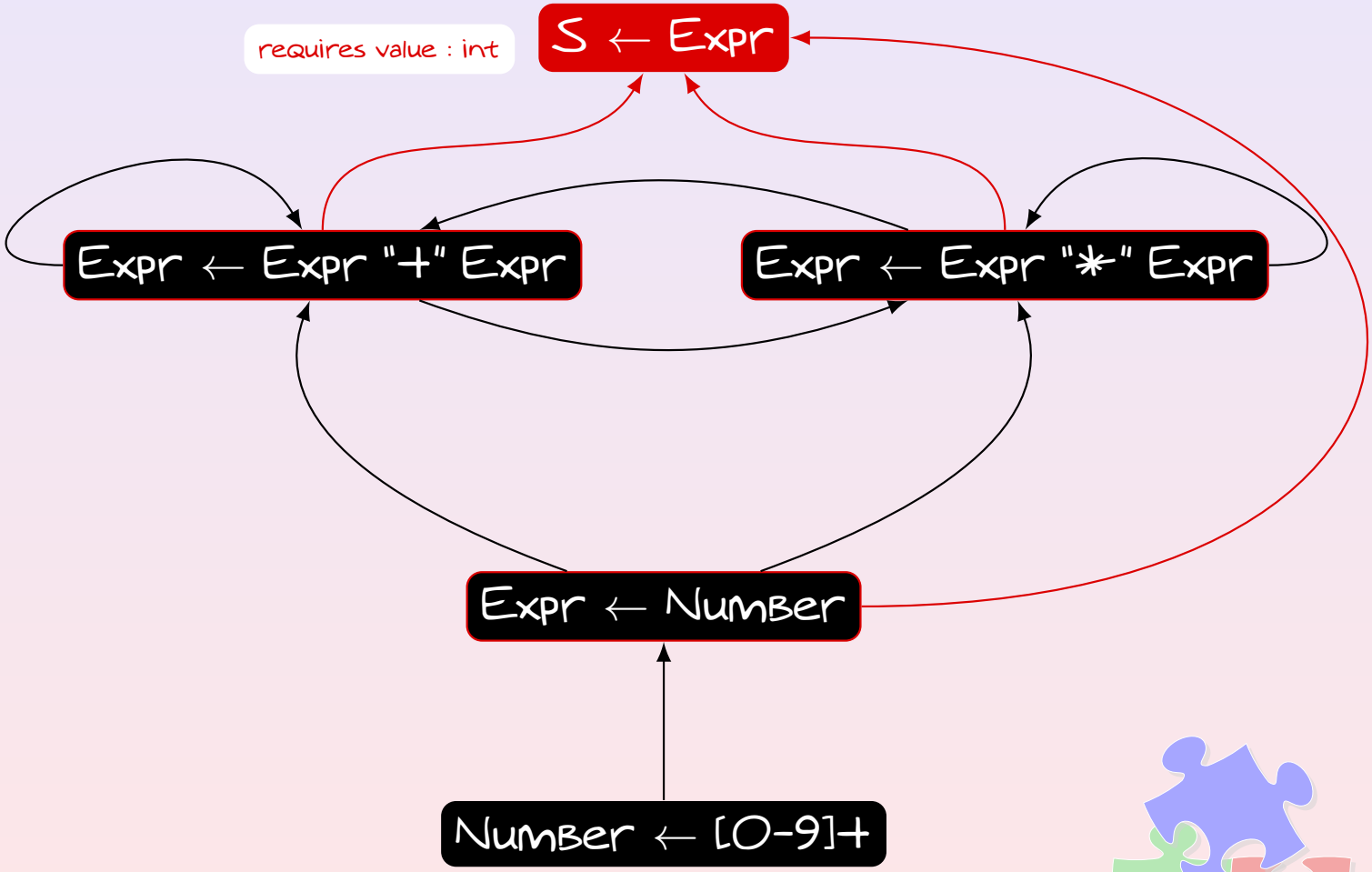
Configure the language semantics

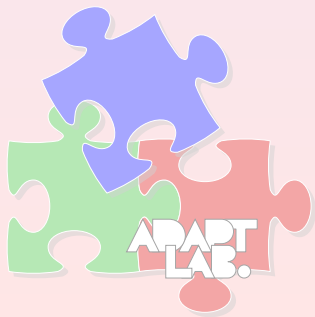




Bottom-up Language Product Line Engineering

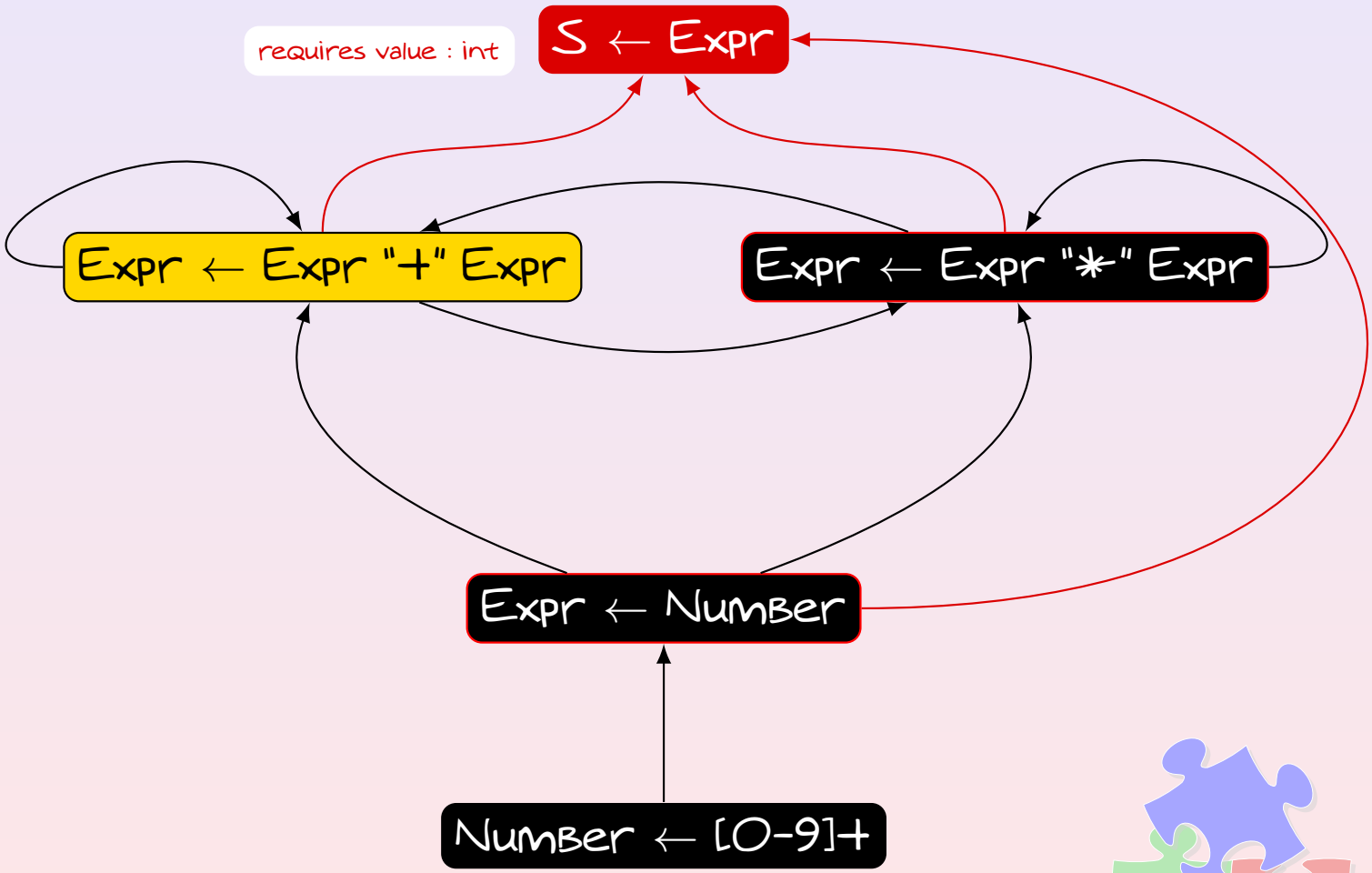
Configure the language semantics

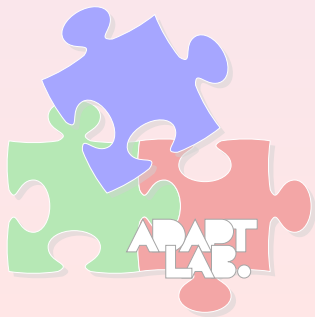




Bottom-up Language Product Line Engineering

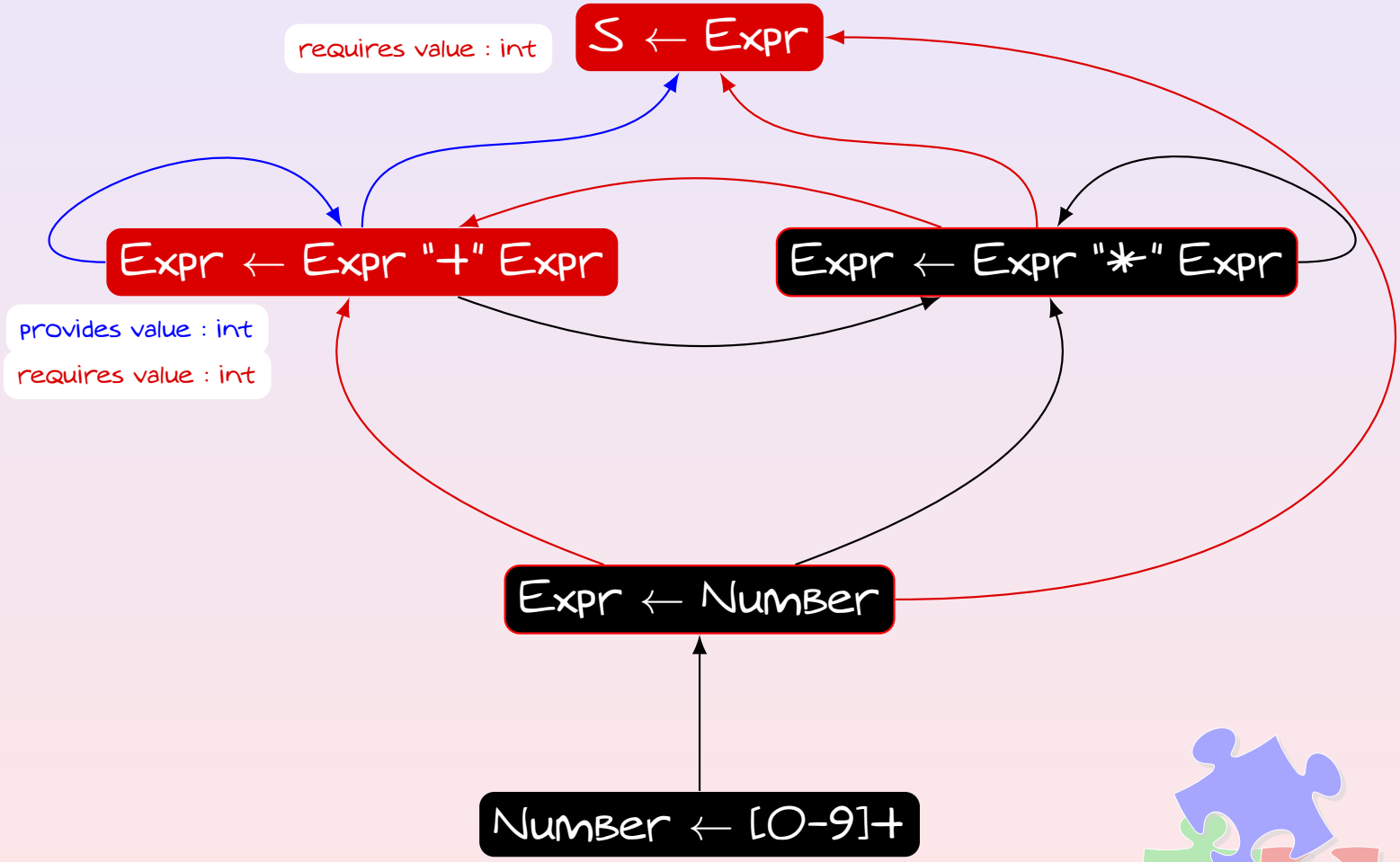
Configure the language semantics

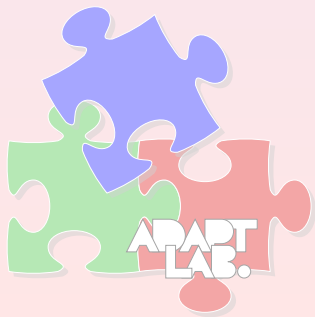




Bottom-up Language Product Line Engineering

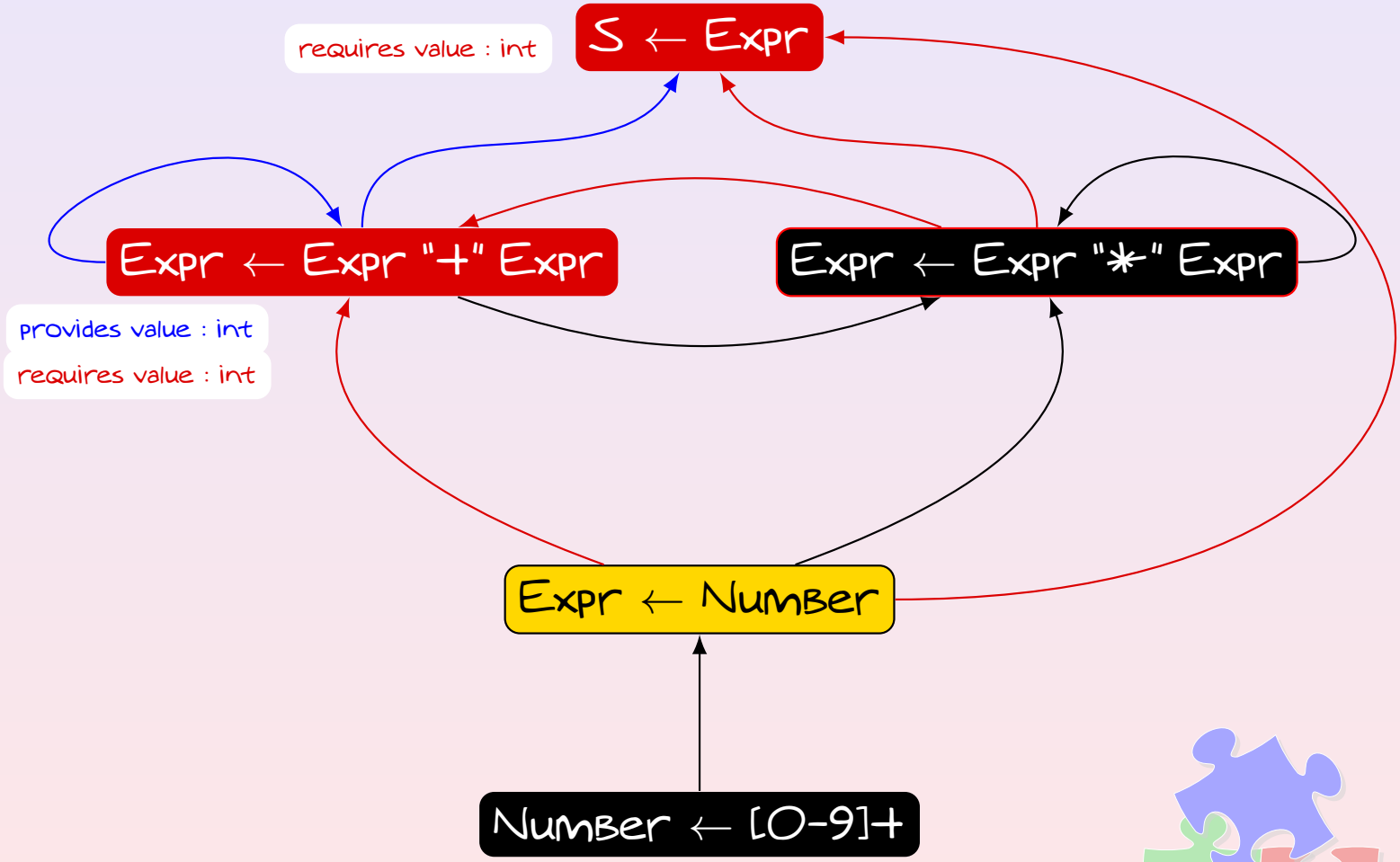
Configure the language semantics

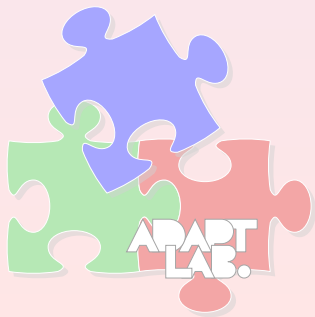




Bottom-up Language Product Line Engineering

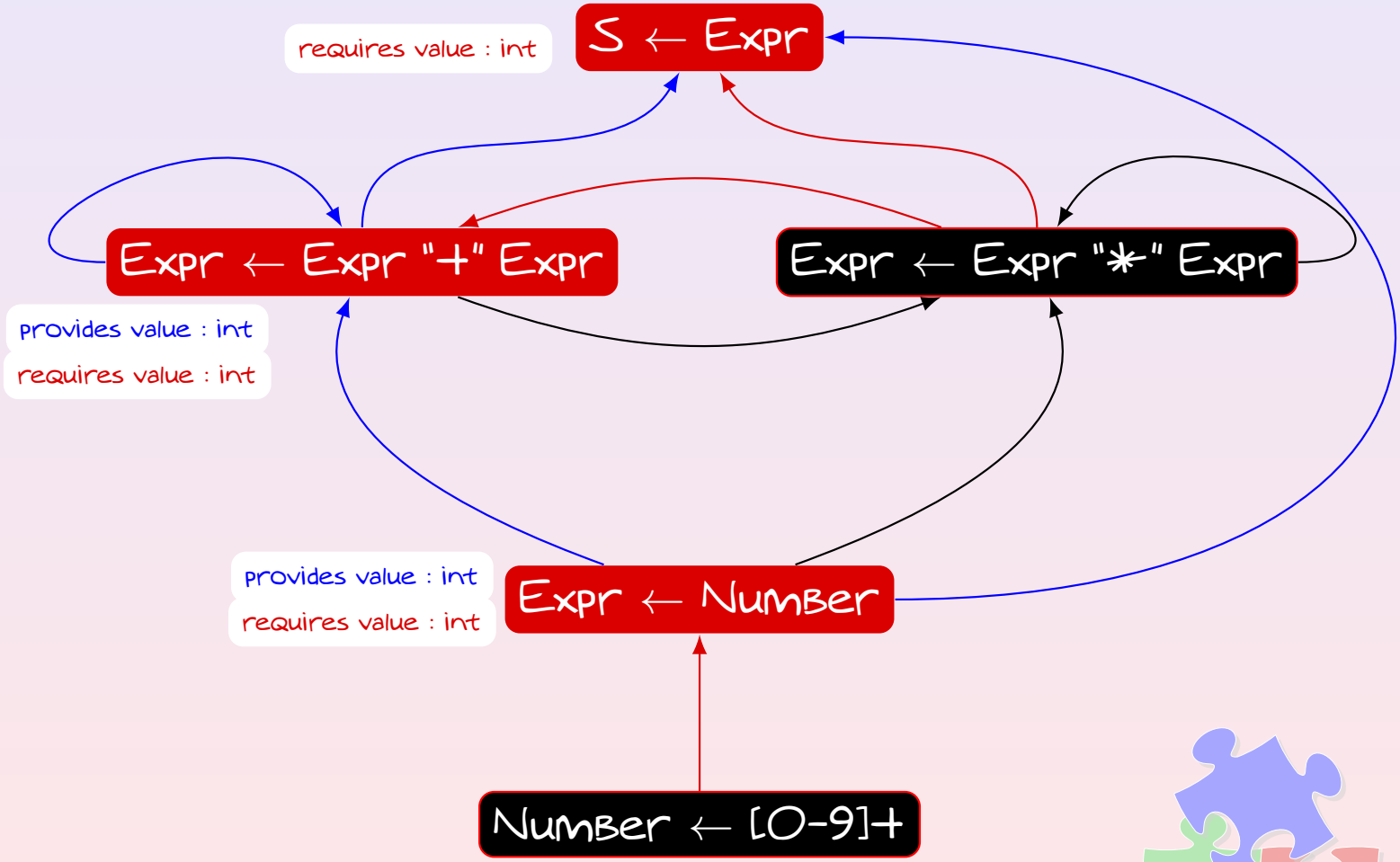
Configure the language semantics

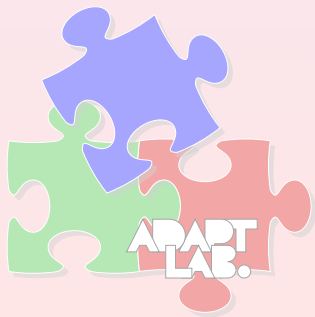




Bottom-up Language Product Line Engineering

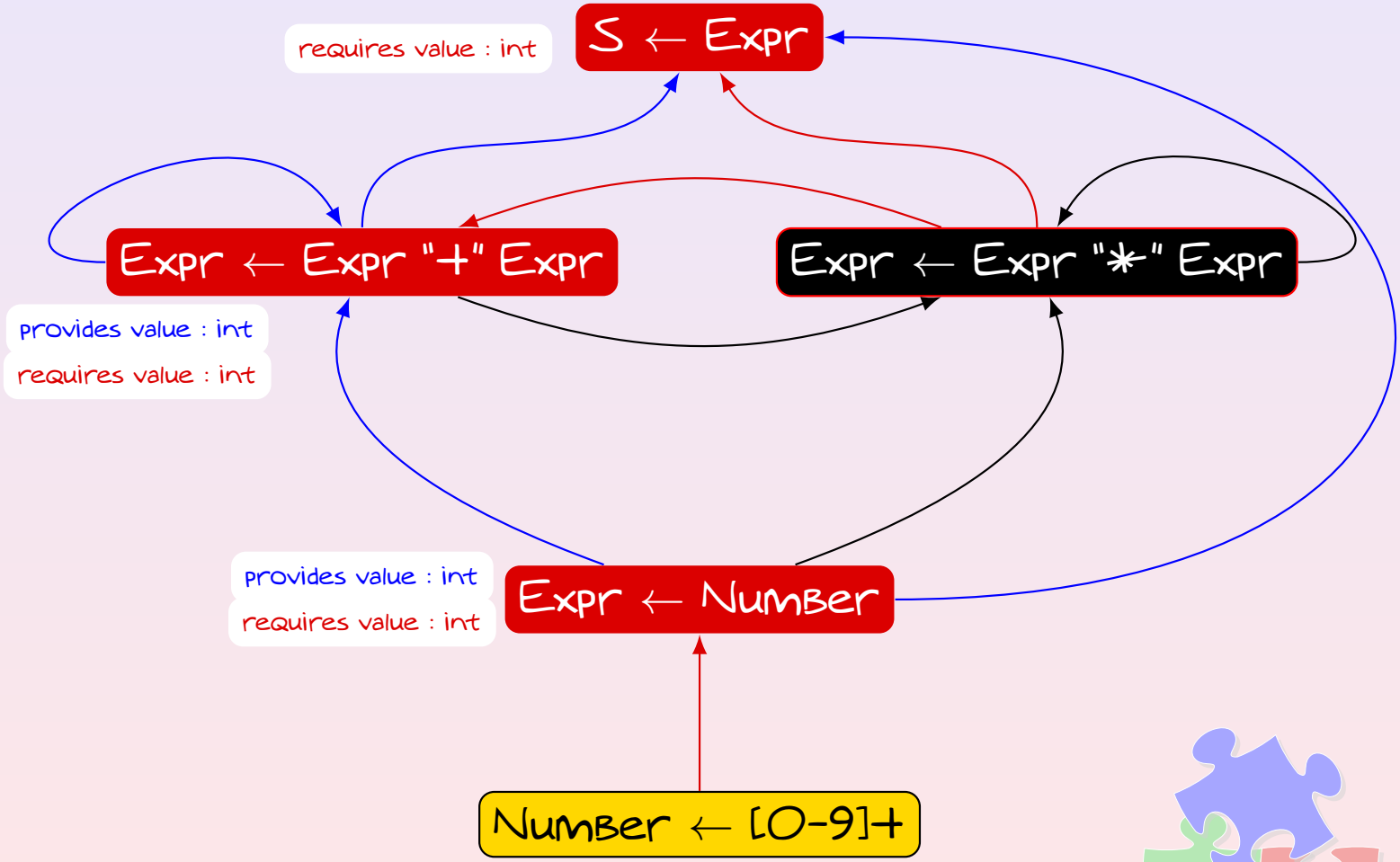
Configure the language semantics

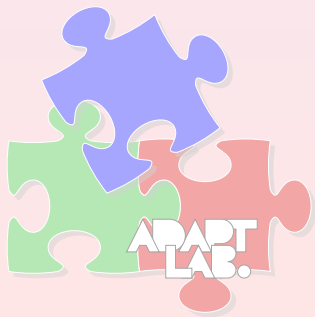




Bottom-up Language Product Line Engineering

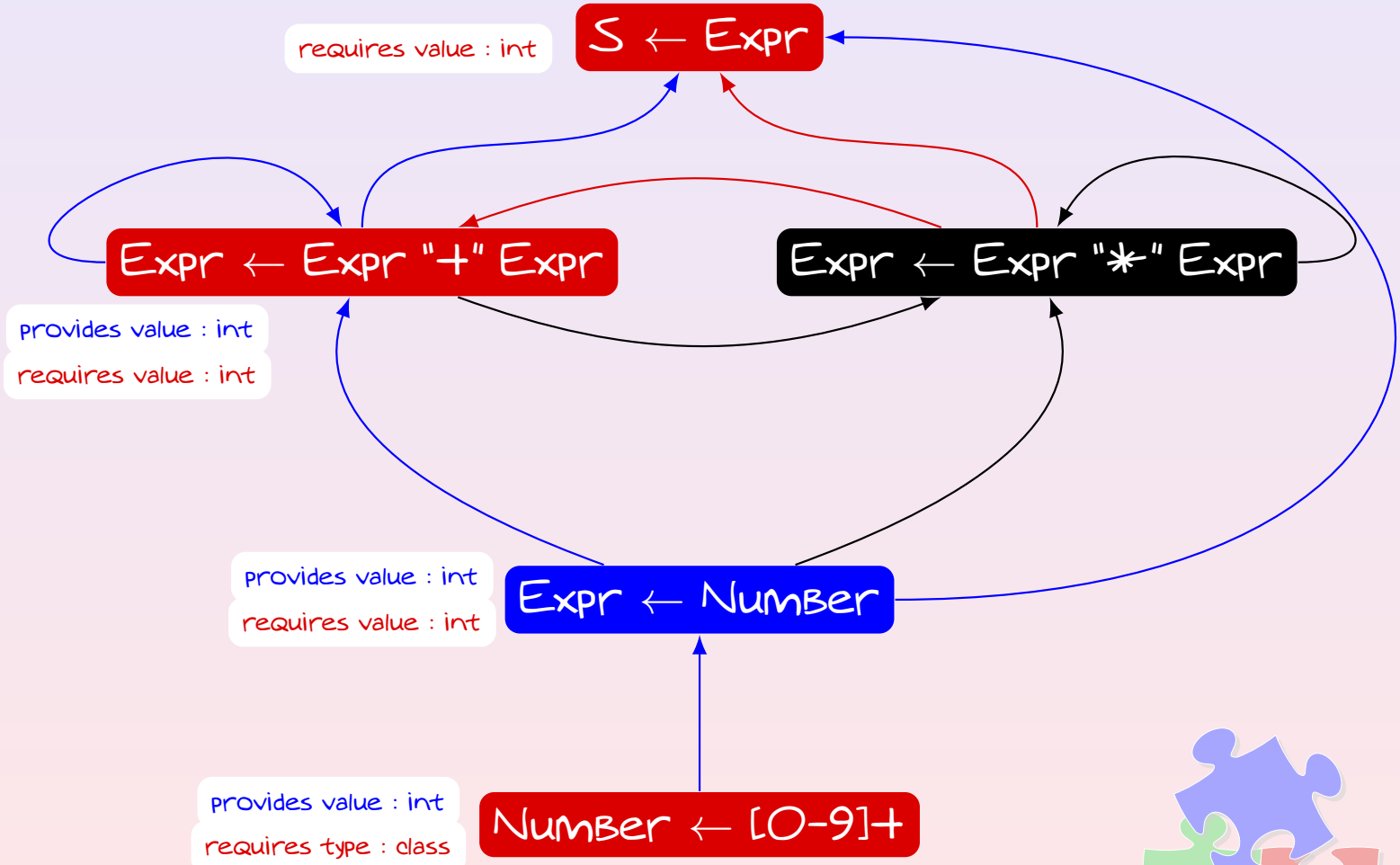
Configure the language semantics

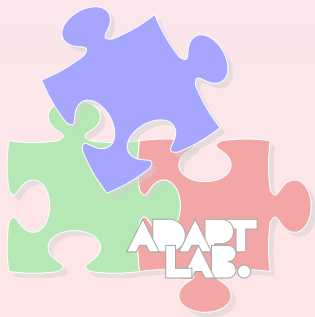




Bottom-up Language Product Line Engineering

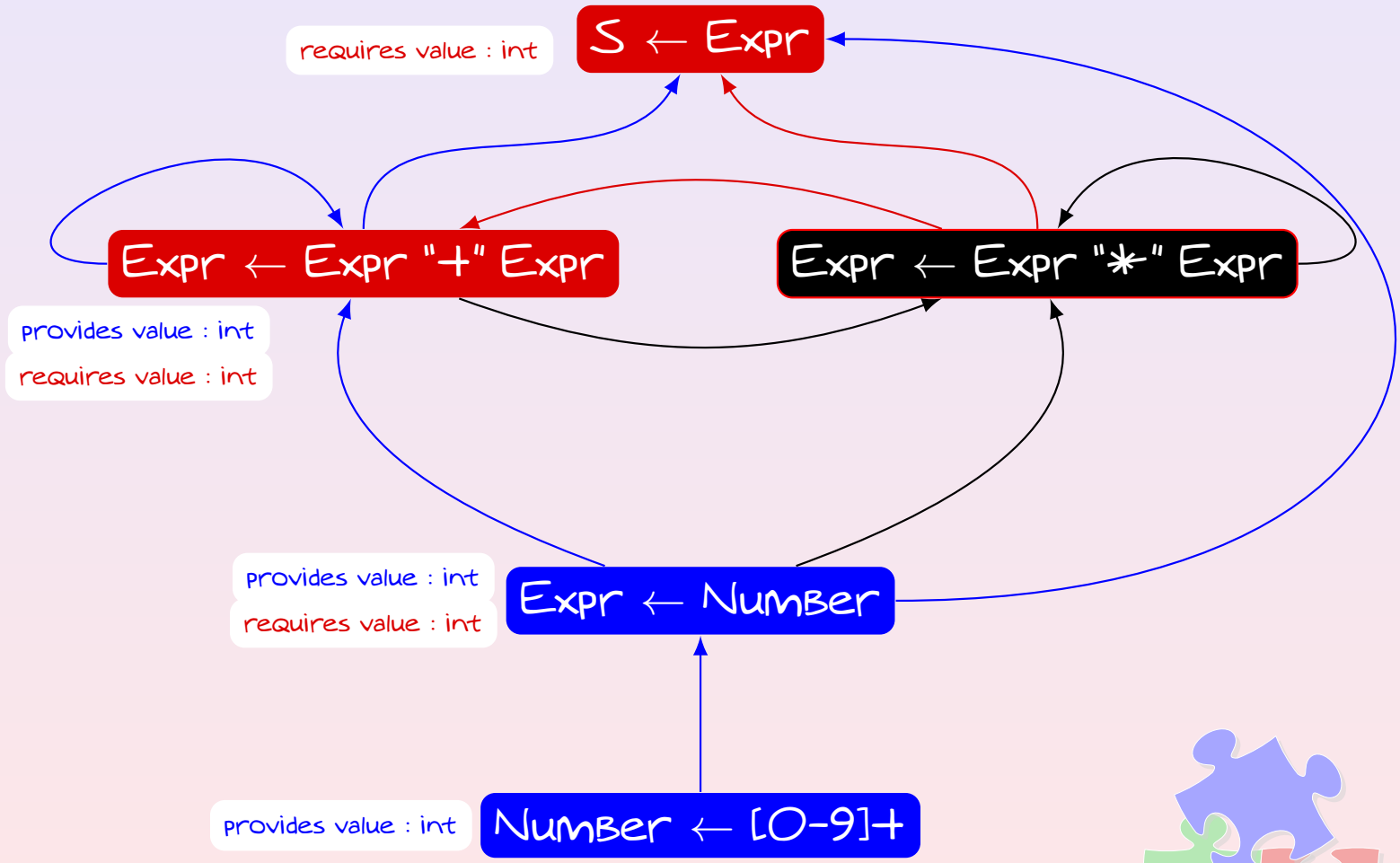
Configure the language semantics

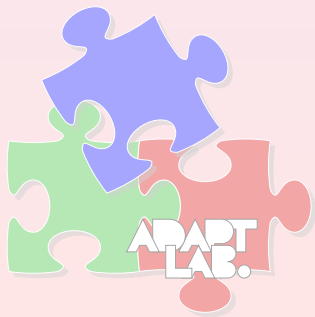




Bottom-up Language Product Line Engineering

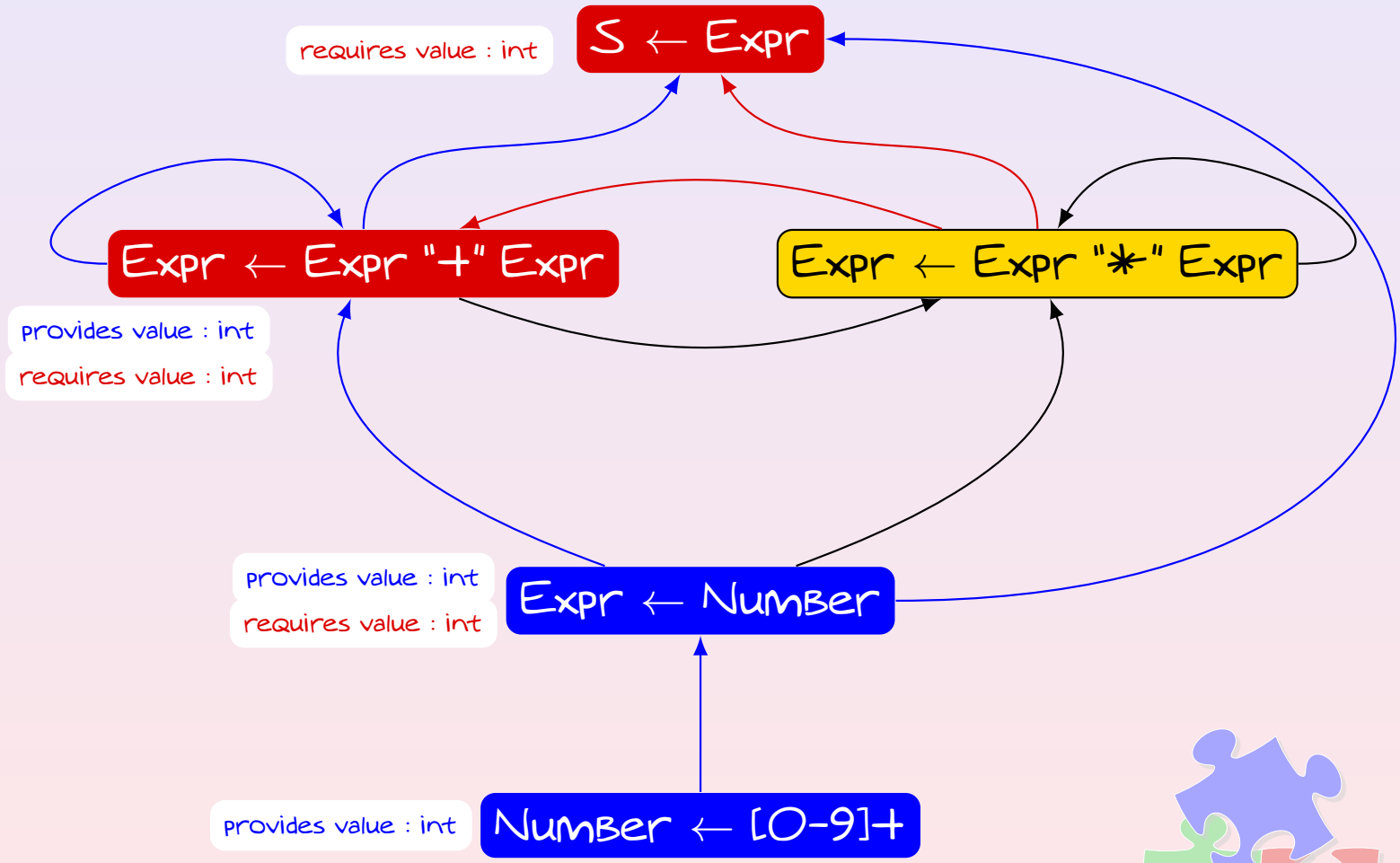
Configure the language semantics

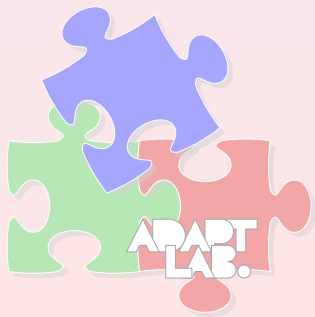




Bottom-up Language Product Line Engineering

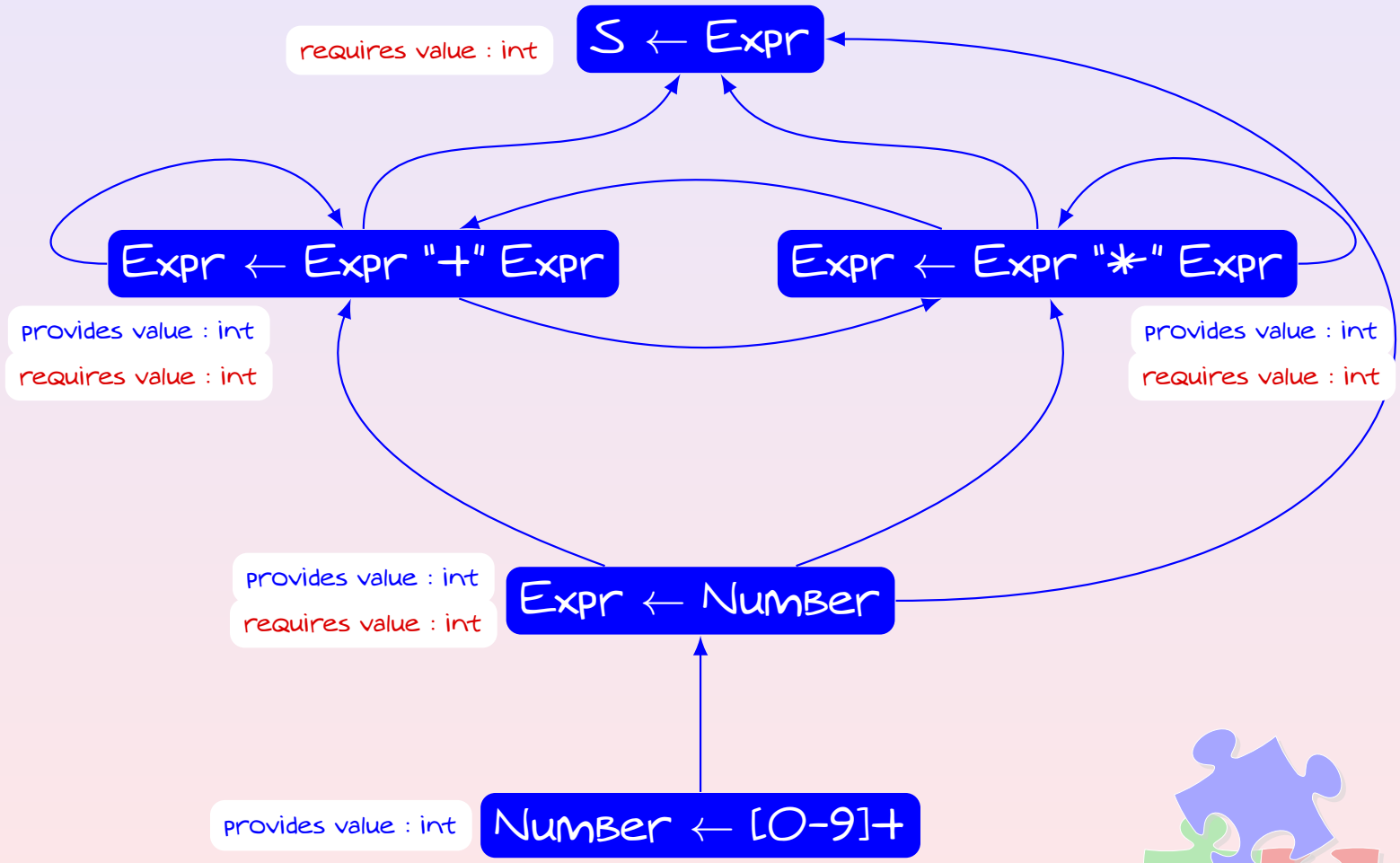
Configure the language semantics





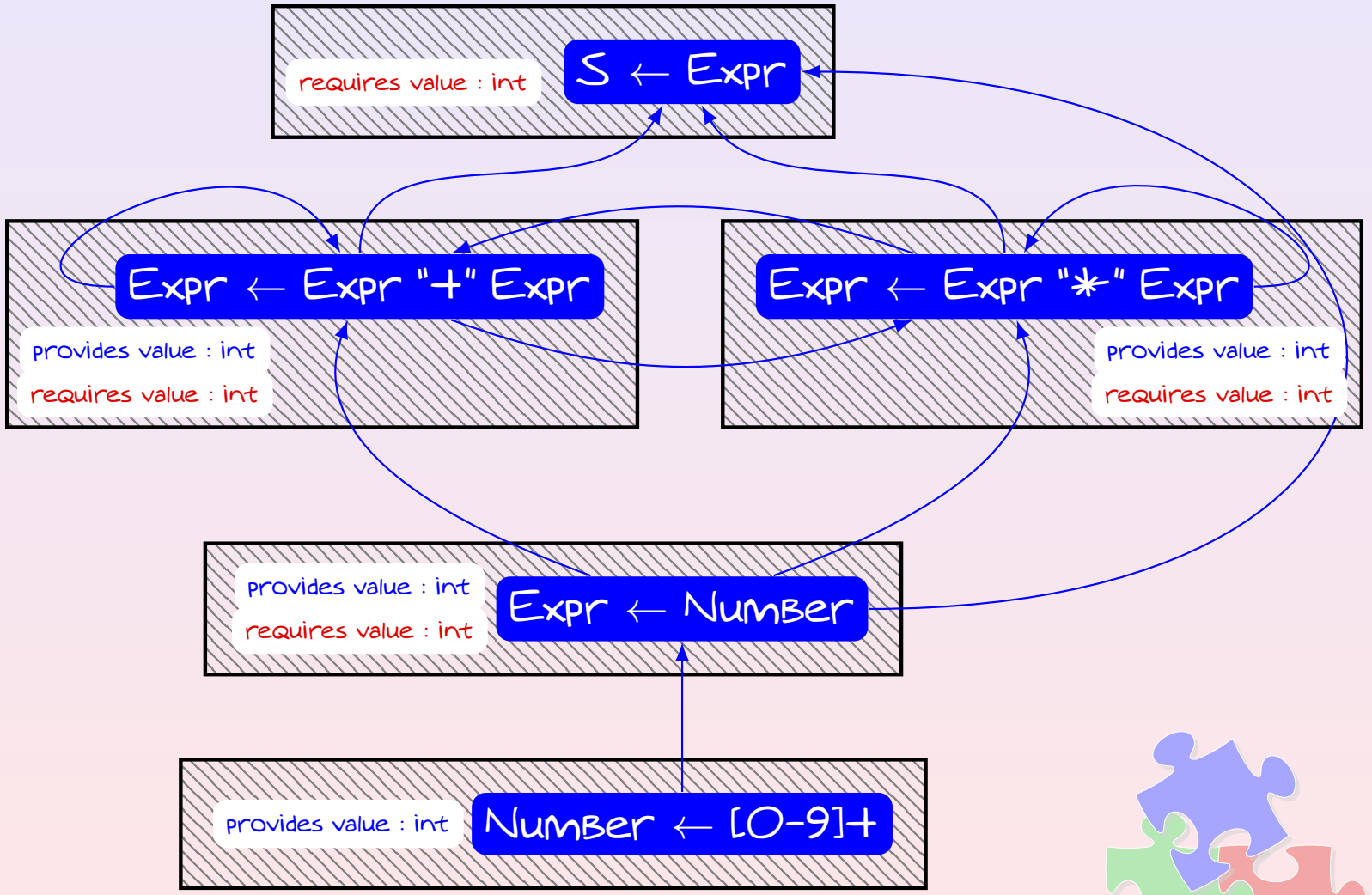
Bottom-up Language Product Line Engineering

Configure the language semantics



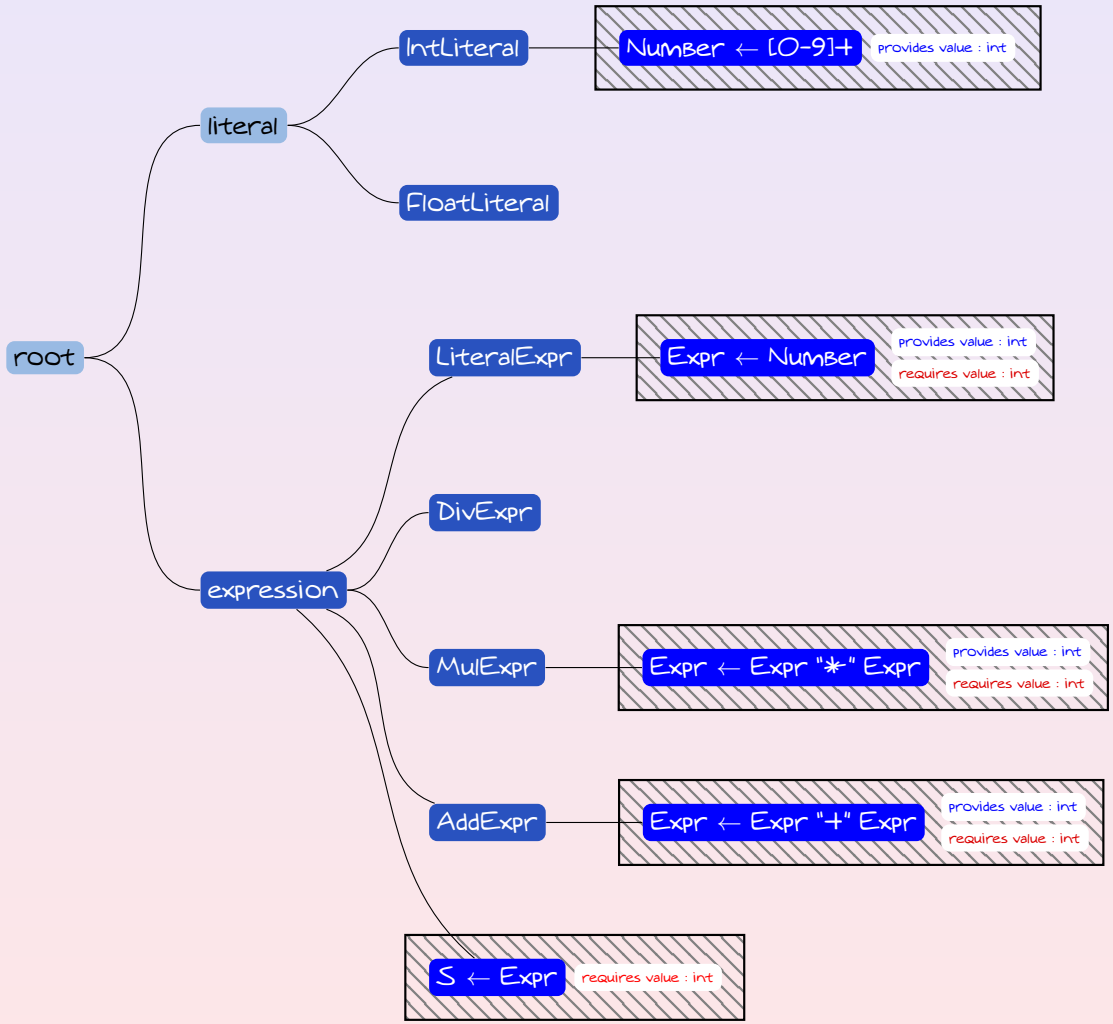
Bottom-up Language Product Line Engineering

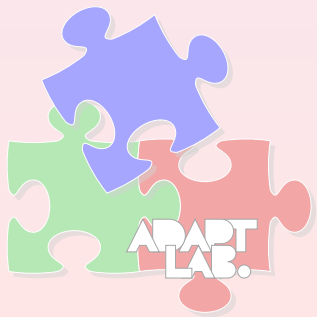
Compose abstract syntax, concrete syntax and semantics into a language feature



Bottom-up Language Product Line Engineering

Increment the variability model with the new language features

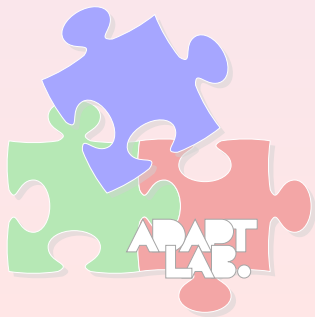




Conclusion

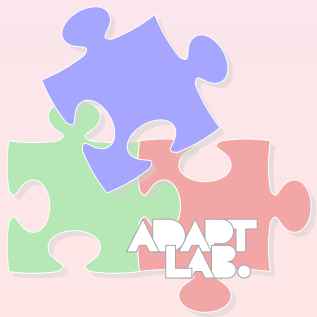
Configuring language variants becomes increasingly simple:

- On the first iteration, the configuration is performed from scratch
- Each time a new valid language feature is created, it is added to the variability model
- Using previously configured language features reduces if not skips at all the semantics resolution stage
- The semantics configuration is driven by the syntax and by pre-conditions and post-conditions to ensure the validity of the final product



The End

Questions ⇐ MayBe Answers



The End

Questions ⇜ MayBe Answers

. . . ⇜ time for your input!