Composing Programming Languages

Navid Roux

2020-01-22

seminar of the kwarc research group

at Friedrich-Alexander-Universtiät Erlangen-Nürnberg

Necessity

- Many GPLs and DSLs available
- Almost all projects use ≥ 2 languages
- How to combine them? [Voe13]
  - Homogeneous / Heterogeneous Fragments

    HTML + CSS files vs. HTML + inline CSS
  - Language Dependent / Independent

    CSS references HTML tags, so that's dependent

What needs to be composed?

[Voe13]

- Syntax
- Static Semantics (constraints, type system)
- Execution Semantics
- IDE Features

(Non-)Examples of Compositions

XML in Scala

```
stocks match {
  case <stocks>{stocks @ _*}</stocks> =>
    for (stock @ <stock>{_*}</stock> <- stocks)
      println(s"stock: ${stock.text}")
}
```

- Syntax

  good

- Static Semantics

  good

- Execution Semantics

  good, via scala.xml.*

- IDE Features

  lacking in IntelliJ IDEA + Scala plugin

HTML in PHP


```
<?php
/* PHP stuff */
?>
<textarea><?php echo $_GET['message']; ?></textarea>
```

- Syntax

  trivial
- Static Semantics

  trivial
- Execution Semantics

  trivial
- IDE Features

  mostly good: delegation to HTML or PHP handlers

Embedded SQL in C

```
int main() {
    int x; char *y; float z;
    EXEC SQL INSERT INTO highscore(id, name, score)
        VALUES(:x, :y, :z);
}
```

- Syntax

  good

- Static Semantics

  good (typechecked)

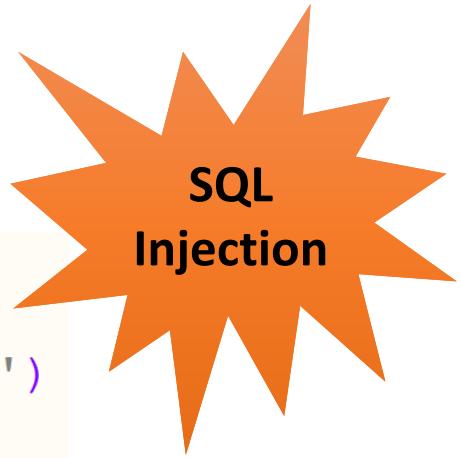
- Execution Semantics

  good

- IDE Features

  lacking in most editors

SQL in almost any language




```php
<?php

$db->query('SELECT * FROM users WHERE pw="' . $pw . '"')

?>
```

- Syntax

  trivial

- Static Semantics

  trivial

- Execution Semantics

  trivial

- IDE Features

  lacking in most editors

Prepared SQL statements in any languag


```php
<?php

$stmt->prepare('SELECT * FROM users WHERE name LIKE ?')
     ->bind('s', '%' . $_GET['pattern'] . '%')
     ->exec();
```

- Syntax — trivial
- Static Semantics — trivial
- Execution Semantics — better than before, but not exhaustive
- IDE Features — lacking in most editors

XPath and Regex in almost any language

```
// XPath
XPathExpression expr = xpath.compile(
  "//pizza[@topping='" + selectedTopping + "']"
)

// Prepared XPath
xpath.setXPathVariableResolver(/* ... */);
XPathExpression expr = xpath.compile(
  "//pizza[@topping=$topping]"
)

// Regex
Pattern regex = Pattern.compile("Pizza \d+ with " + selectedTop
```

Java's javax.xml.xpath lib allows variables!

Conclusion

Even a single language can have many sublanguages

Naive compositions

- easy to implement
- correlate with vulnerabilities

Ideal compositions

- hard to implement
- statically typechecked
- provide runtime semantics

Excursion: Composing Logics

Why are logic compositions relatively easier?

Overview

See also [Rab17, Mül19]

- Syntax

Inclusion

```
theory T = {include Logic1, include Logic2}
```

- Static Semantics (~~constraints,~~ type system)

Inclusion

- Execution Semantics

Given `v: Logic1 ⟶ ZF`, `w: Logic2 ⟶ ZF`

build `u: T ⟶ ZF = {include v, include w}`

- IDE Features

Inclusion?

Why is it so easy? Because...

- Syntax

…we work directly on AST

- Static Semantics (constraints, type system)

…we specify a flexible type system
via inheritable rules

- Execution Semantics

…we specify semantics in terms of rewriting

- IDE Features

Actually not easy ☺

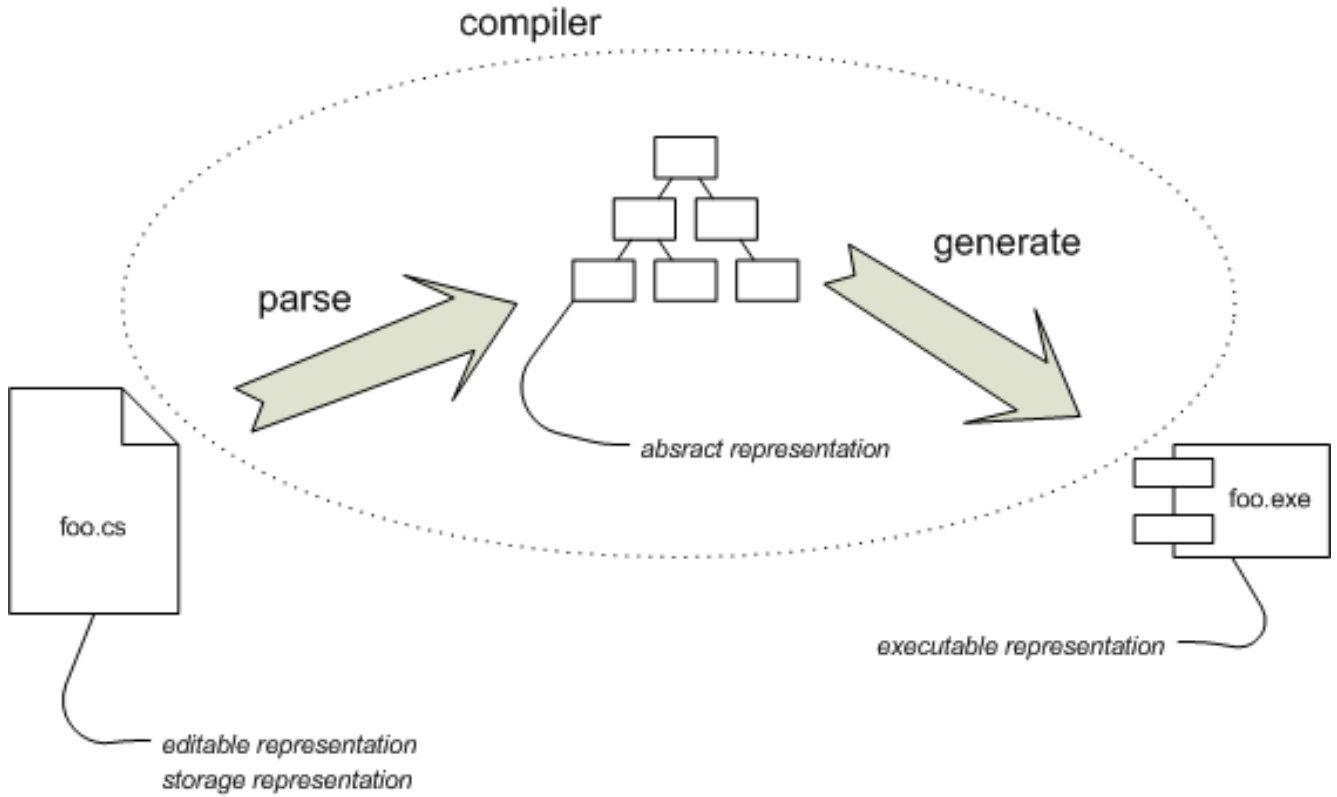
Editors & Concepts

Editors for Composing Languages

- Multiple available, see [Wik20, Voe13]
- Focus on JetBrains MPS & concepts here

Syntax Composition

Textual Editing



[Fow05b]

- Default among programmers
- Needs parser
- Allows ambiguity and syntax errors
  e.g. `int class = template + 5` is invalid C++

⇨ Not well-suited for composition!

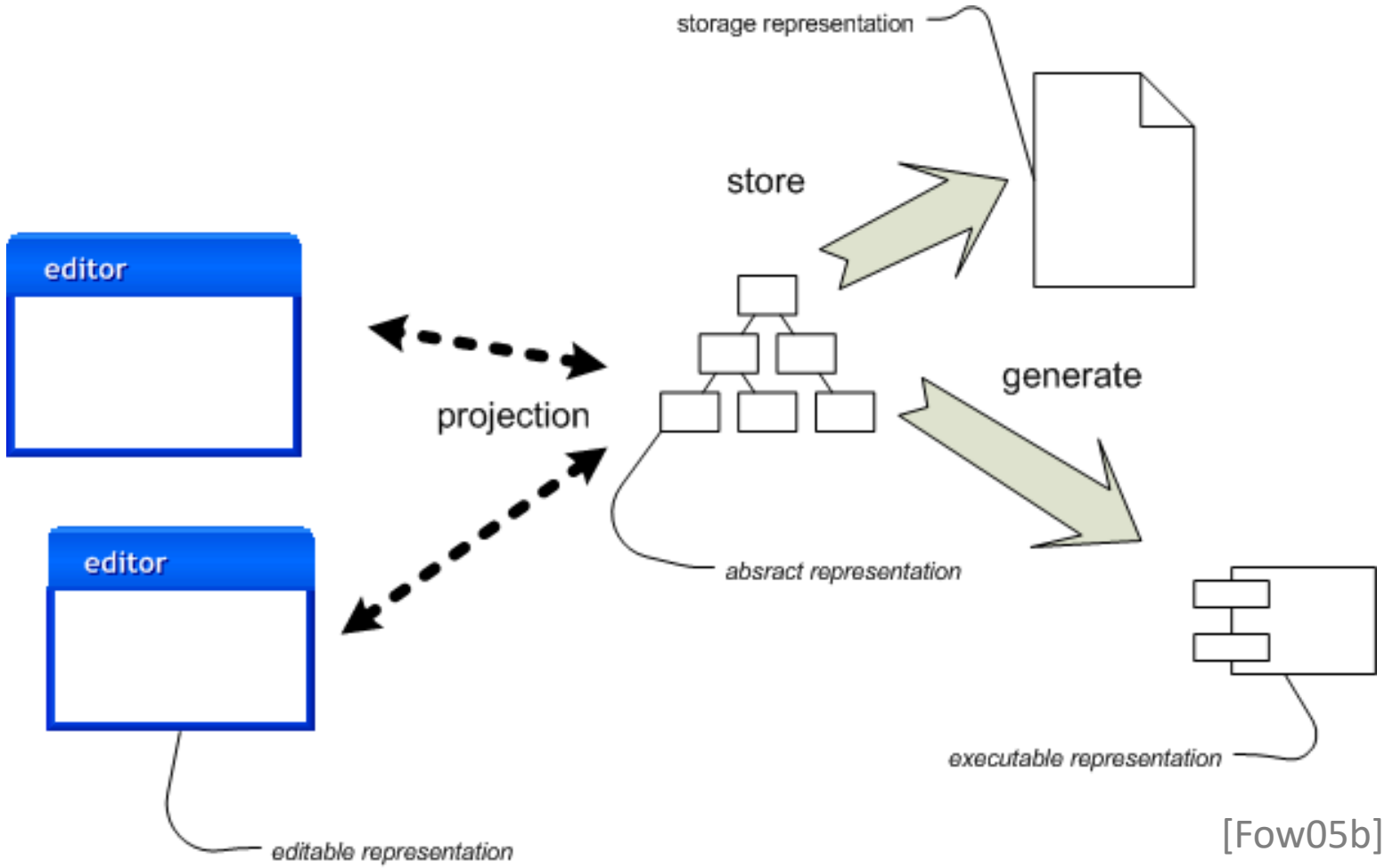
Structural/Projectional Editing

- We all know this
  - Word
  - PowerPoint
  - Scratch
- All work is done on AST
- No ambiguity and syntax errors

  `int class = template + 5` is representable

⇨ Use this for compositional editing!

Structural/Projectional Editing (cont.)

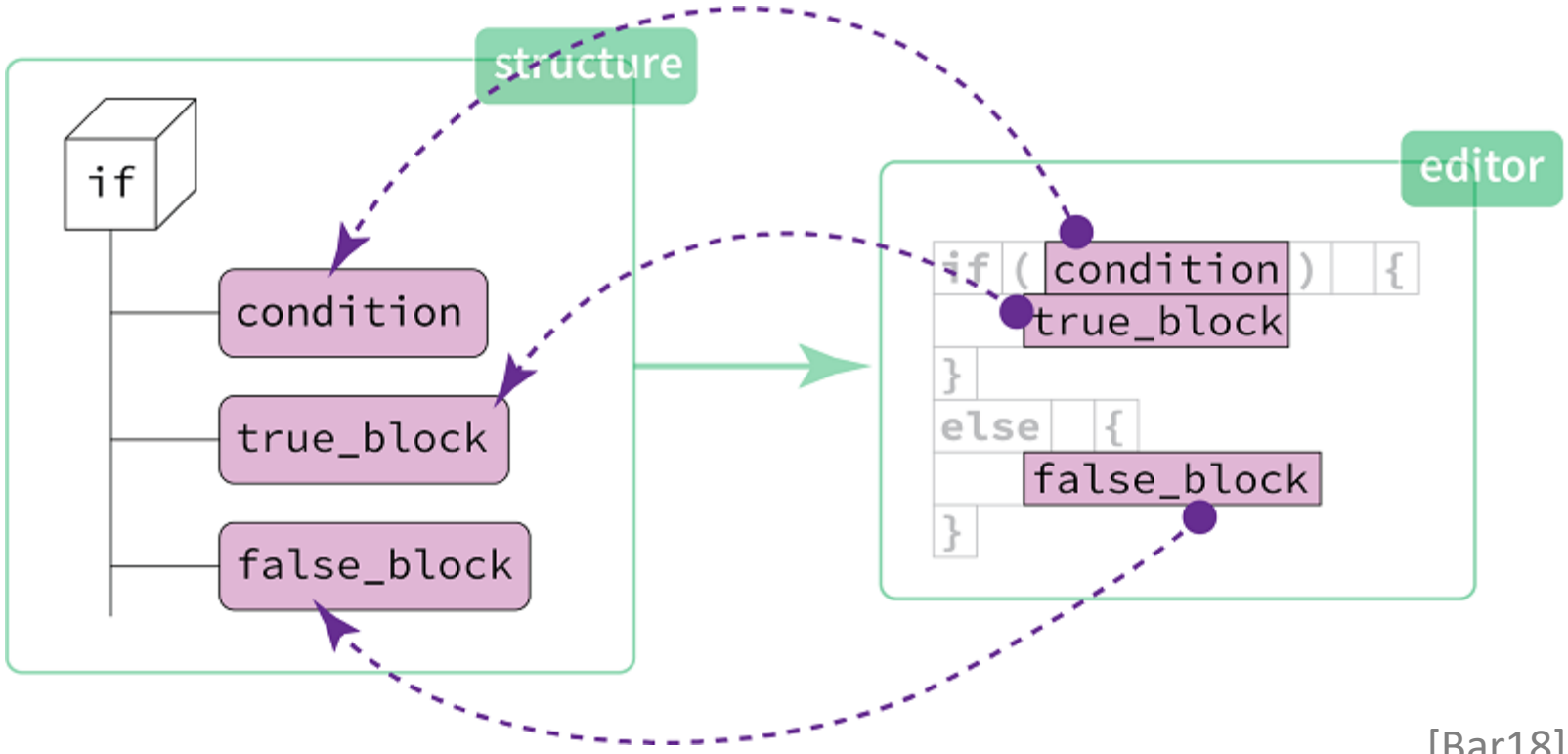


[Fow05b]

Live Demo:
Projectional Editing in MPS

by means of the „complexLanguage“ built-in example in MPS

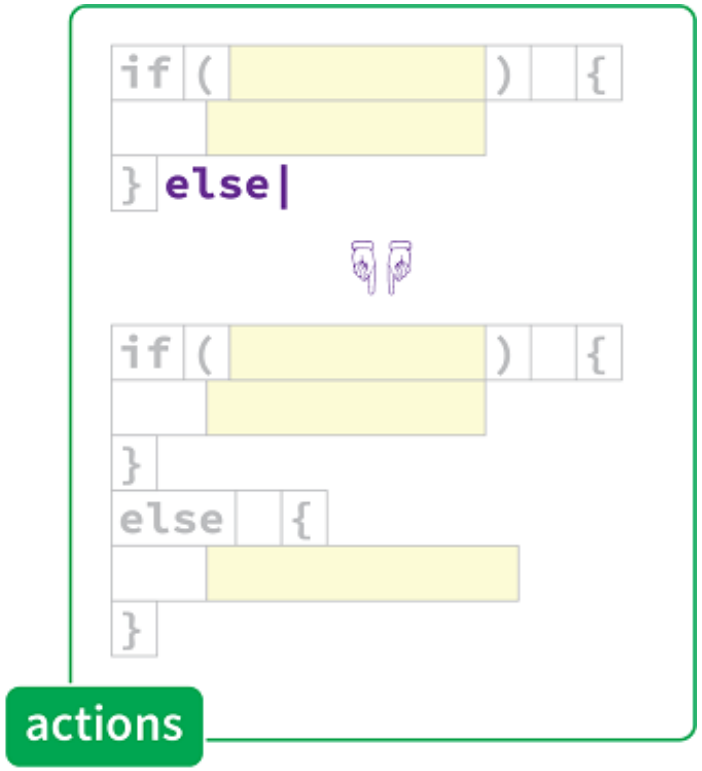
MPS Editors



[Bar18]

Declare editors for every concept

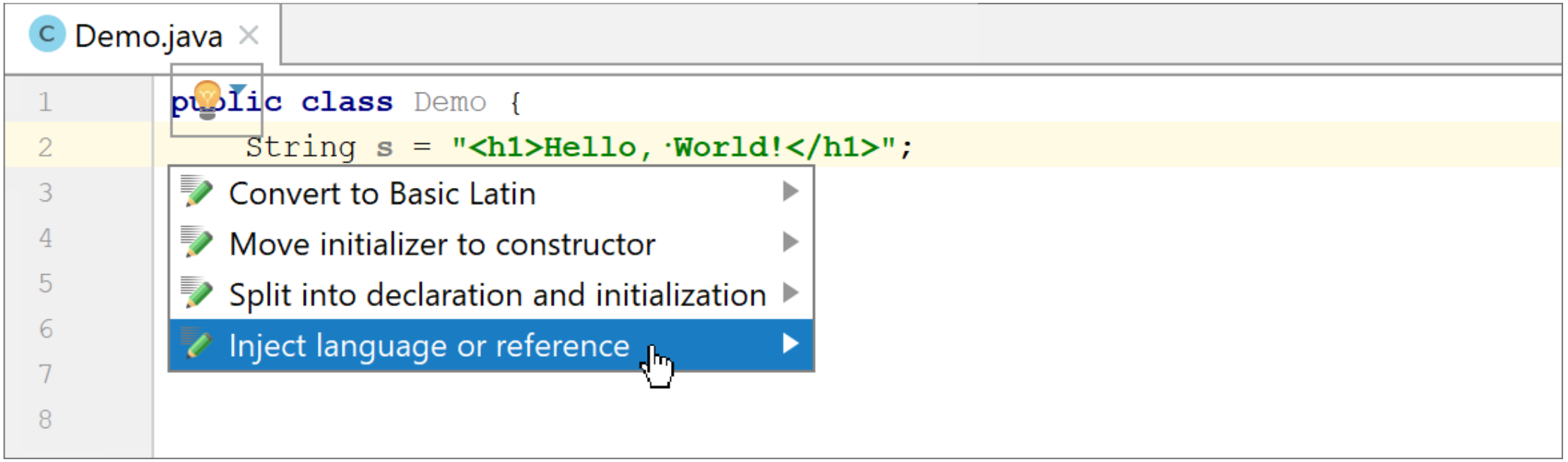
MPS Actions



[Bar18]

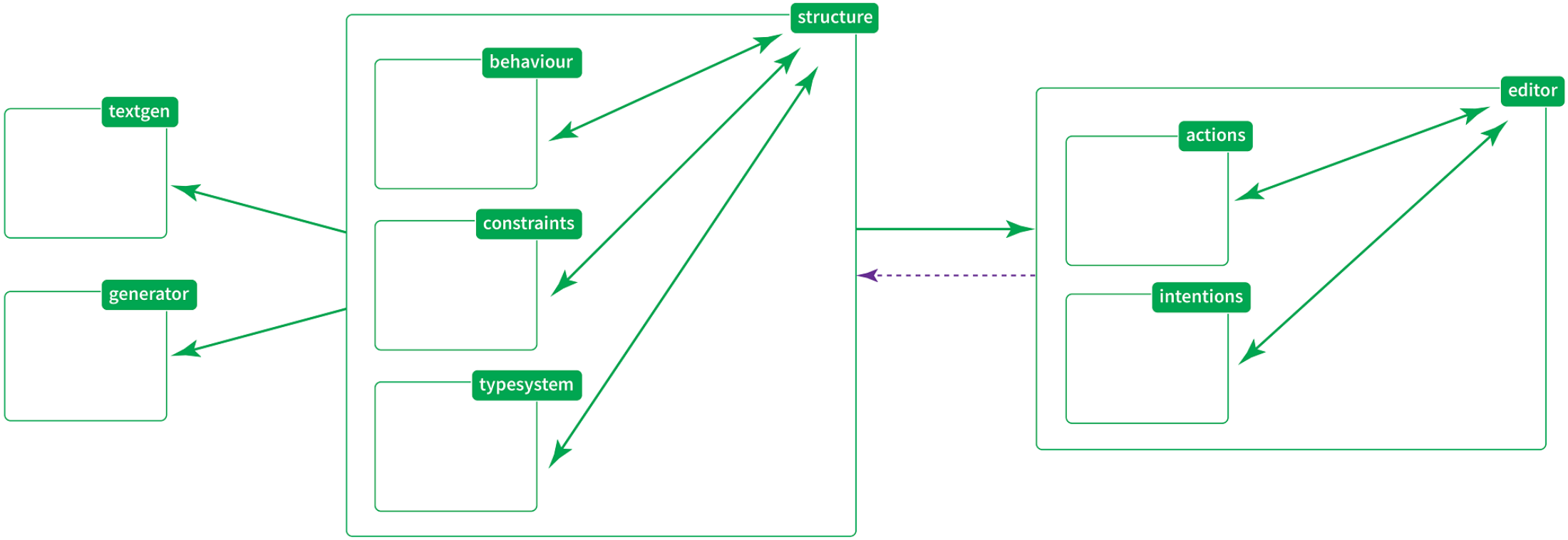
Declare actions to blur the line
between textual & projectional editing

MPS Intentions



Declare intentions (aka quickfixes)

MPS Overview



[Bar18]

Semantic Composition

Constraint & Type System

"MPS' type system specification is based on declarative typing rules that are executed by a solver." [Voe13]

- Every language can define its own rules
- The rule sets get mixed in the composition

⇨ Typing rules very similar to MMT

What about constraints?

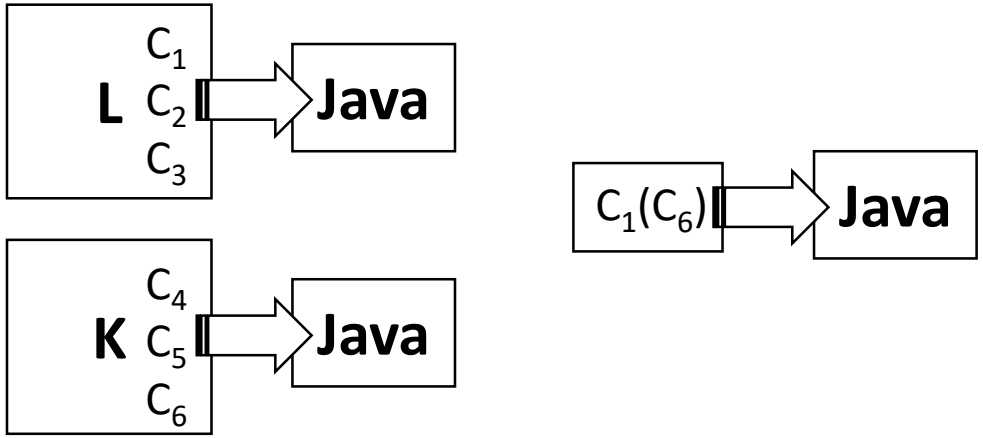
Live Demo:
Constraint & Type System in MPS

by means of the „lambda“ built-in example in MPS

Execution Semantics

Modular Transformation Framework

- Choose target language for execution semantics
- Provide transformations for every concept in every language



- Induces homomorphic transformation on (L ∪ K)
- That can be overridden by rules for special cases of composed syntax

Final Comparison

Comparison

MMT

- Syntax:
  - mostly works on AST
  - offers (ambiguous) notation
- Static Semantics
  - Purely additive via rules
- Execution Semantics
  - Purely additive via views & rules
- IDE
  - Text-based notational editor

JetBrains MPS

- Syntax
  - works on AST
  - offers custom editors
- Static Semantics
  - Additive & constraint-based
- Execution Semantics
  - Additive & overriding-based
- IDE
  - Editors, Actions and Intentions

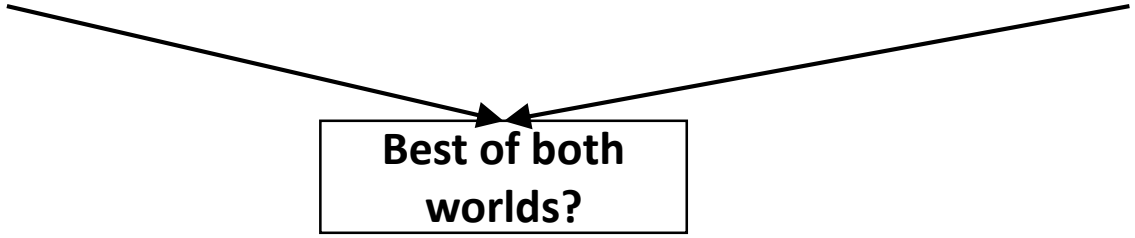
Comparison (cont.)

MMT

- Large fine-grained API
- API accessible via Scala (a GPL!)
- Extensible via structural features (e.g. diag. operators)
- Formal system

JetBrains MPS

- ?
- ?
- Potentially via new concept and generator? How to cache/optimize?
- Language Workbench [Fow05a]



Bibliography (MPS & soft. eng. side)

- [Bar18]: Barash, Mikhail. “Looking at Code through the Prism of JetBrains MPS,” August 6, 2018. https://medium.com/@mikhail.barash.mikbar/looking-at-code-through-the-prism-of-jetbrains-mps-8e9b70e3257d.
- [Voe13]: Voelter, Markus. “Language and IDE Modularization and Composition with MPS.” In *Generative and Transformational Techniques in Software Engineering IV*, edited by Ralf Lämmel, João Saraiva, and Joost Visser, 7680:383–430. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. https://doi.org/10.1007/978-3-642-35992-7_11.
- [Fow05a] Fowler, Martin. “Language Workbenches: The Killer-App for Domain Specific Languages?” *MartinFowler.Com* (blog), June 12, 2005. https://www.martinfowler.com/articles/languageWorkbench.html.
- [Fow05b] Fowler, Martin. “ProjectionalEditing.” *MartinFowler.Com* (blog), January 14, 2008. https://martinfowler.com/bliki/ProjectionalEditing.html.
- [Wik20]: Wikipedia contributors. *Language Workbench — Wikipedia, The Free Encyclopedia*, 2019. https://en.wikipedia.org/w/index.php?title=Language_workbench&oldid=904003896.

Bibliography (MMT & logic side)

- [Rab17] Rabe, Florian. “How to Identify, Translate, and Combine Logics?” *Journal of Logic and Computation* 27, no. 6 (2017): 1753–1798.
- [Mül19] Müller, Dennis. “Mathematical Knowledge Management Across Formal Libraries.” PhD Thesis, Informatics, FAU Erlangen-Nürnberg, 2019. http://gl.kwarc.info/supervision/PhD-archive/mueller_dennis/dmueller_phd.pdf.