# MPS Coderules: Constraint programming for type inference

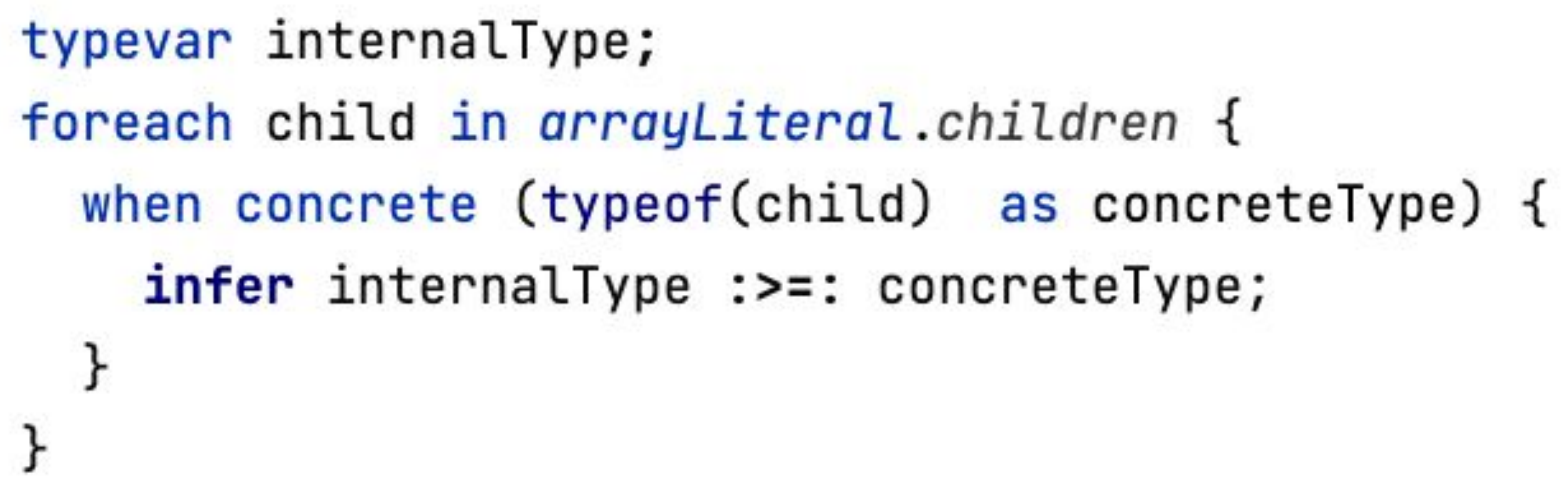
Clément de La Bourdonnaye

Aachen - Sept 2022

## Presentation outline

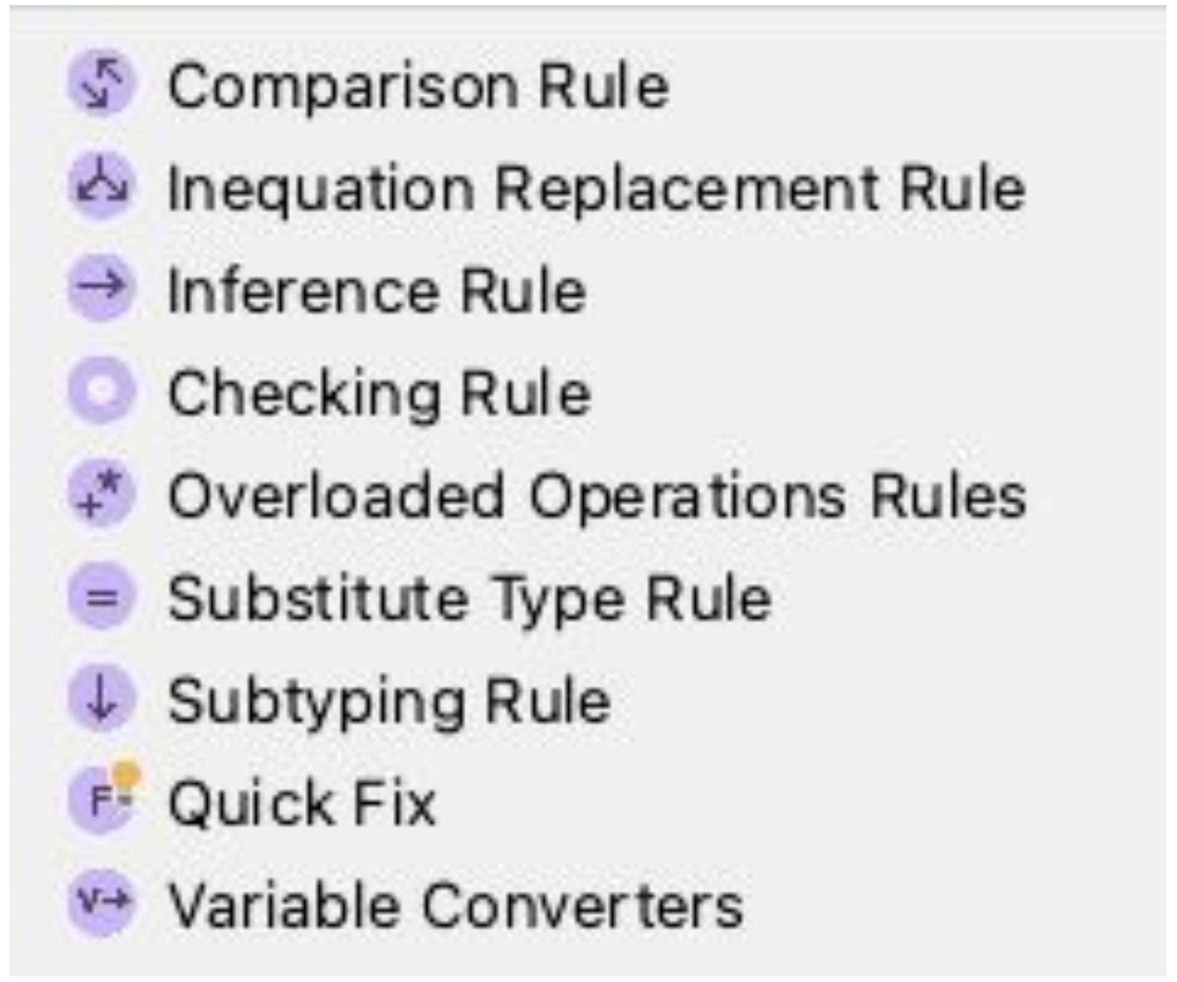
- Why Coderules?
- Constraint programming basics
- Demo: constraints program as type system
- MPS Kotlin and Coderules implementation

## Why Coderules?



```
typevar internalType;
foreach child in arrayLiteral.children {
  when concrete (typeof(child)  as concreteType) {
    infer internalType :>=: concreteType;
  }
}
```

- ➔ Legacy typesystem definition shortcomings
  - ◆ Pre-defined instructions
    - No customization
    - Precise behavior hard to grasp
  - ◆ Complex language features impossible to express

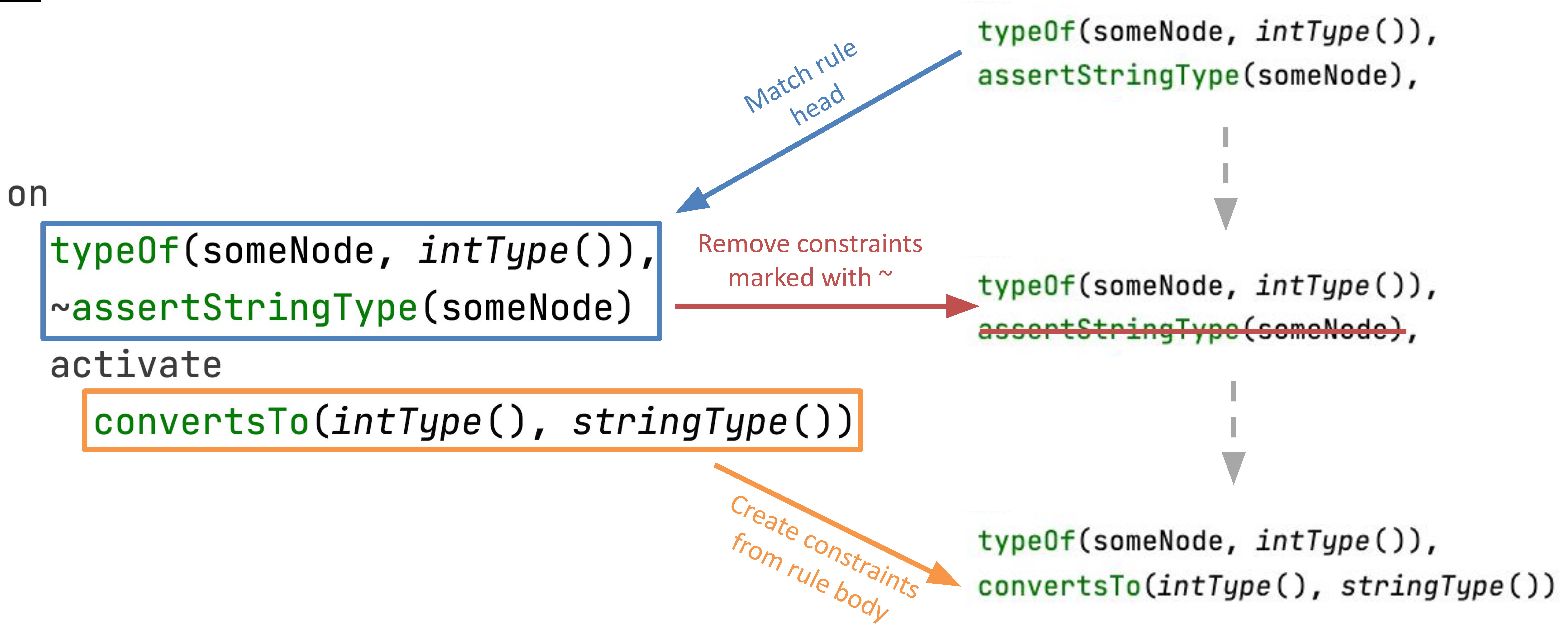


Comparison Rule
Inequation Replacement Rule
Inference Rule
Checking Rule
Overloaded Operations Rules
Substitute Type Rule
Subtyping Rule
Quick Fix
Variable Converters

## Constraint?

```
checkAll() / 0
typeOf(what: node<>, itsType: term) / 2
convertsTo(left: term, right: term) / 2
```

## Constraint handling rules



```
on
  typeOf(someNode, intType()),
  ~assertStringType(someNode)
  activate
    convertsTo(intType(), stringType())
```

```
typeOf(someNode, intType()),
assertStringType(someNode),
```

```
typeOf(someNode, intType()),
~~assertStringType(someNode)~~,
```

```
typeOf(someNode, intType()),
convertsTo(intType(), stringType())
```

Match rule head
Remove constraints marked with ~
Create constraints from rule body

## Logical variables and patterns

- Coderules terms
  - Internal structure
  - Can hold children variables
- Logical variables
  - Placeholder for value unknown yet
  - Can be used now, assigned later
- Patterns
  - Complex matching for rules head

```
open primType (
  <no features>
)

intType : primType (
  value val
)
```

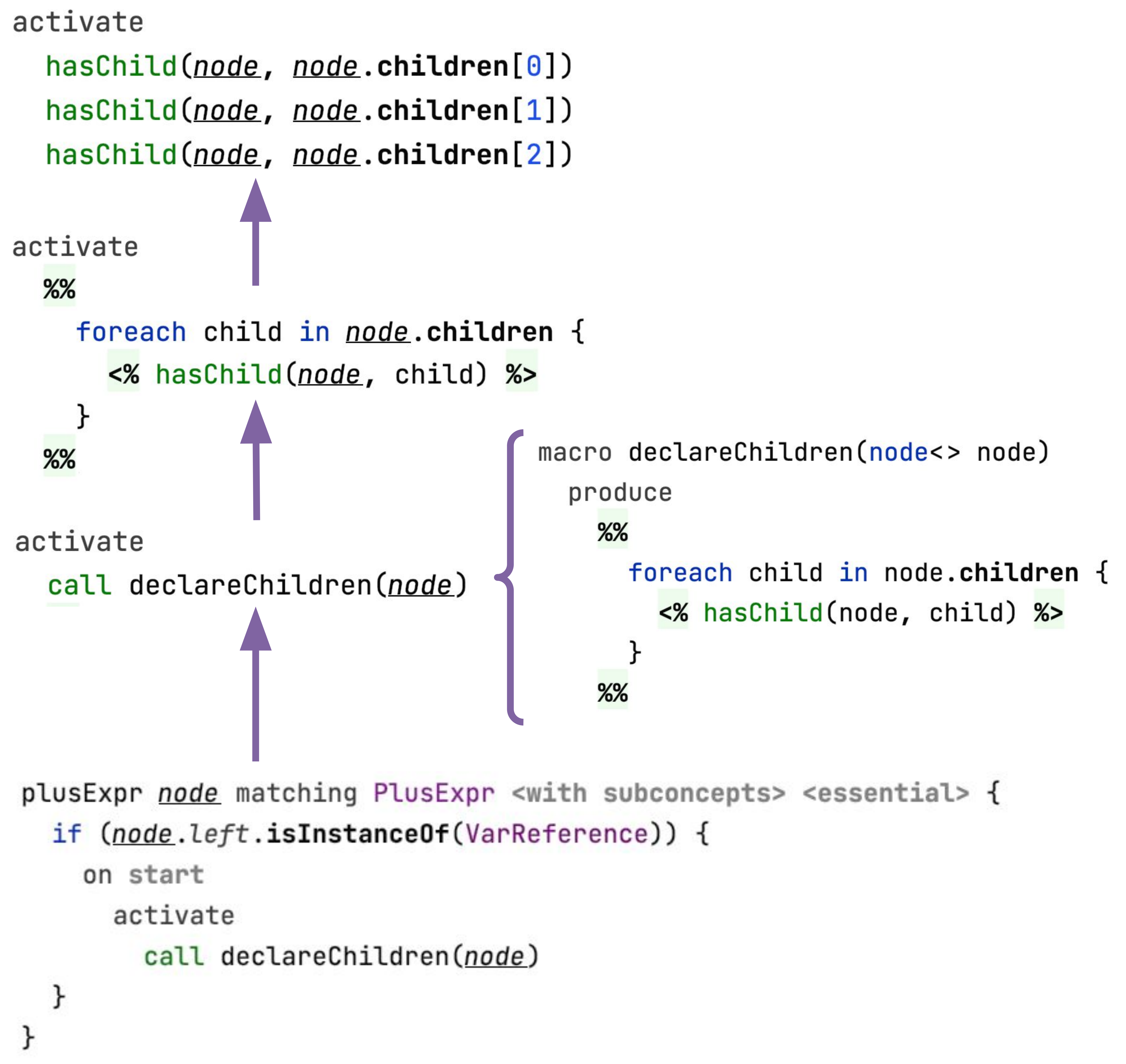
```
convertsTo(TypeVar, intType())
TypeVar = stringType()
                ↓
convertsTo(stringType(), intType())
```

```
on <term Left, Right>
  convertsTo(arrayType(of: Left), arrayType(of: Right))
  activate
    convertsTo(Left, Right)
```

## Coderules: augmented rules definition

—

- ➔ Java code
  - ◆ Templates
  - ◆ Evaluations during rule processing
- ➔ Macros
  - ◆ High code reusability
  - ◆ Before rule processing
- ➔ Rules made for MPS nodes
  - ◆ Rules specific to nodes (typing rules, inheritance…)



```
activate
  hasChild(node, node.children[0])
  hasChild(node, node.children[1])
  hasChild(node, node.children[2])
```

```
activate
  %%
    foreach child in node.children {
      <% hasChild(node, child) %>
    }
  %%
```

```
activate
  call declareChildren(node)
```

```
macro declareChildren(node<> node)
  produce
    %%
      foreach child in node.children {
        <% hasChild(node, child) %>
      }
    %%
```

```
plusExpr node matching PlusExpr <with subconcepts> <essential> {
  if (node.left.isInstanceOf(VarReference)) {
    on start
      activate
        call declareChildren(node)
  }
}
```

## Live Demo

Let’s go!

## Conclusion

| Legacy type system | Coderules |
|---|---|
| ➔ Built-in behaviour<br>◆ Limited control<br>◆ Hard to debug or understand<br><br>➔ One format for all languages | ➔ Full control on each aspect of the typesystem<br><br>➔ Readable debugging<br><br>➔ More extensibility and flexibility<br><br>➔ One implementation per language<br>◆ Working on a new language requires learning its internals<br>◆ but easy access to sources |

## Thank you
## for your attention

—

https://sites.google.com/jetbrains.com/mps-coderules-links

jetbrains.com