# JetBrains MPS

## Projectional editing

and its implications

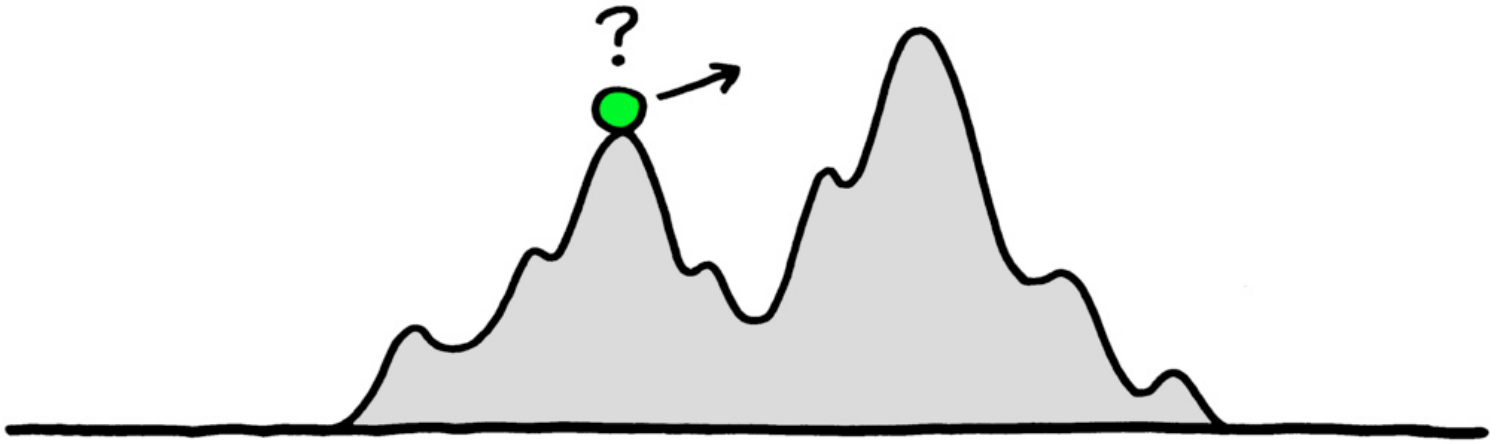
Václav Pech


JET
BRAINS


AC
CL
DC
DM
DP
DT
HB
IJ
YT
PC
PS
R#
R+
RM
TC
UP
WS

Programming had been invented before we were born.

# Local or global optimum?



# MPS - an open-source language workbench for DSLs

# DSLs

Focuse on a narrow problem domain

Raise the level of abstraction

Provide rich set of notations

```
On button: 1  -->  Schedule
On button: 2  -->  Speakers
On button: 3  -->  Practical information
On button: #  -->  Hanging up

Activity: Schedule on button: 1
      On button: 1  -->  Day 1
      On button: 2  -->  Day 2
      On button: 3  -->  Workshops

      Activity: Day 1 on button: 1
            On button: 1  -->  Making It Count
            On button: 2  -->  Cracking the code to secure software
            On button: 3  -->  TestContainers integration testing without the hassle

            Activity: Making It Count on button: 1  Playback Text:
               All too often quality comes along as an afterthought to the software systems we build.
               Get info

            Activity: Cracking the code to secure software on button: 2  Playback Text:
               What is it that makes writing secure software so difficult?
               Get info

            Activity: TestContainers integration testing without the hassle
               on button: 3  Playback Text:
               Unit testing is nice, but without a proper integration testing you might not know how your

               Get info
```

# Tooling

Code-completion

Intentions, Refactoring

Navigation

Searches

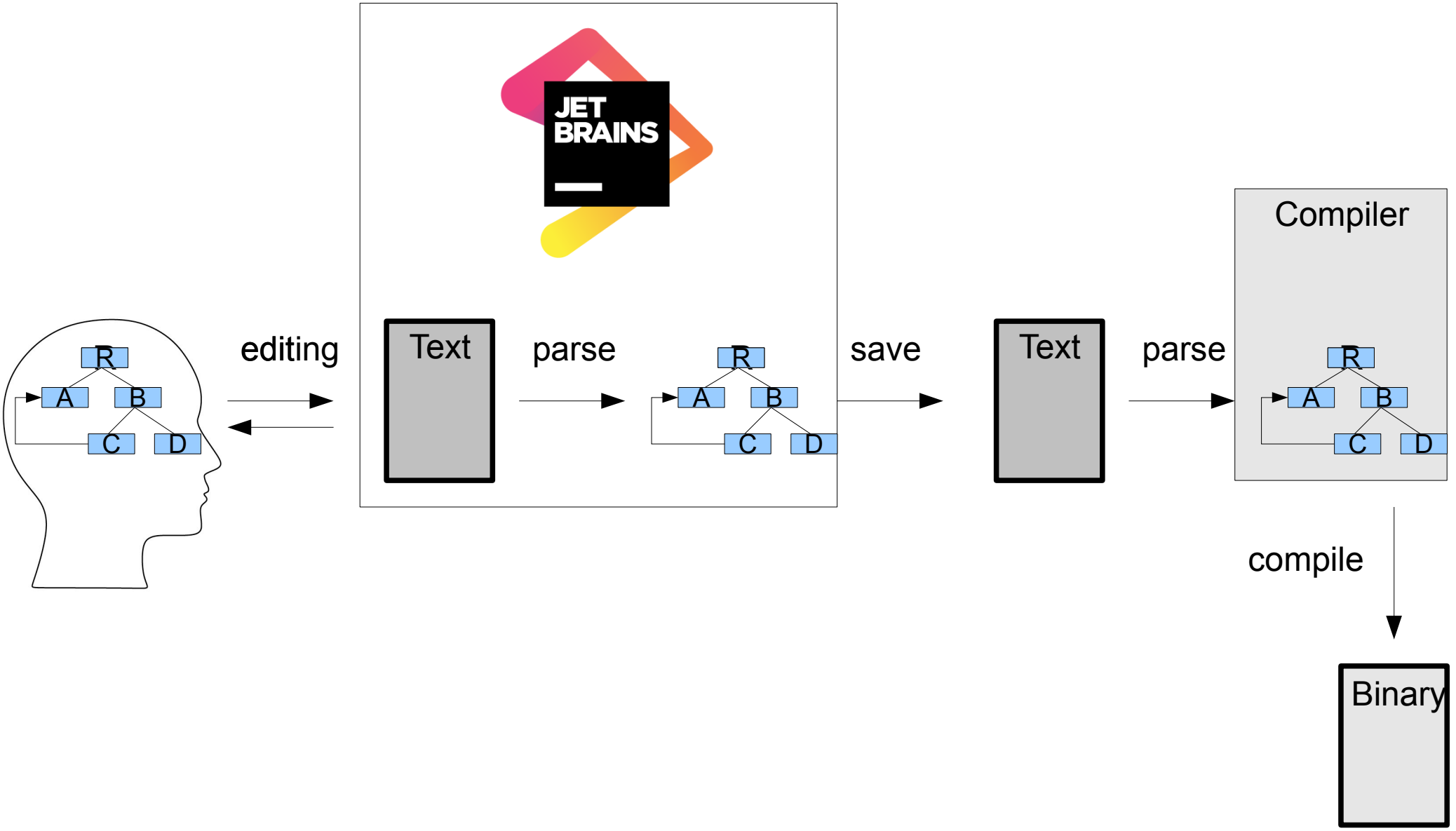
Version control

Testing

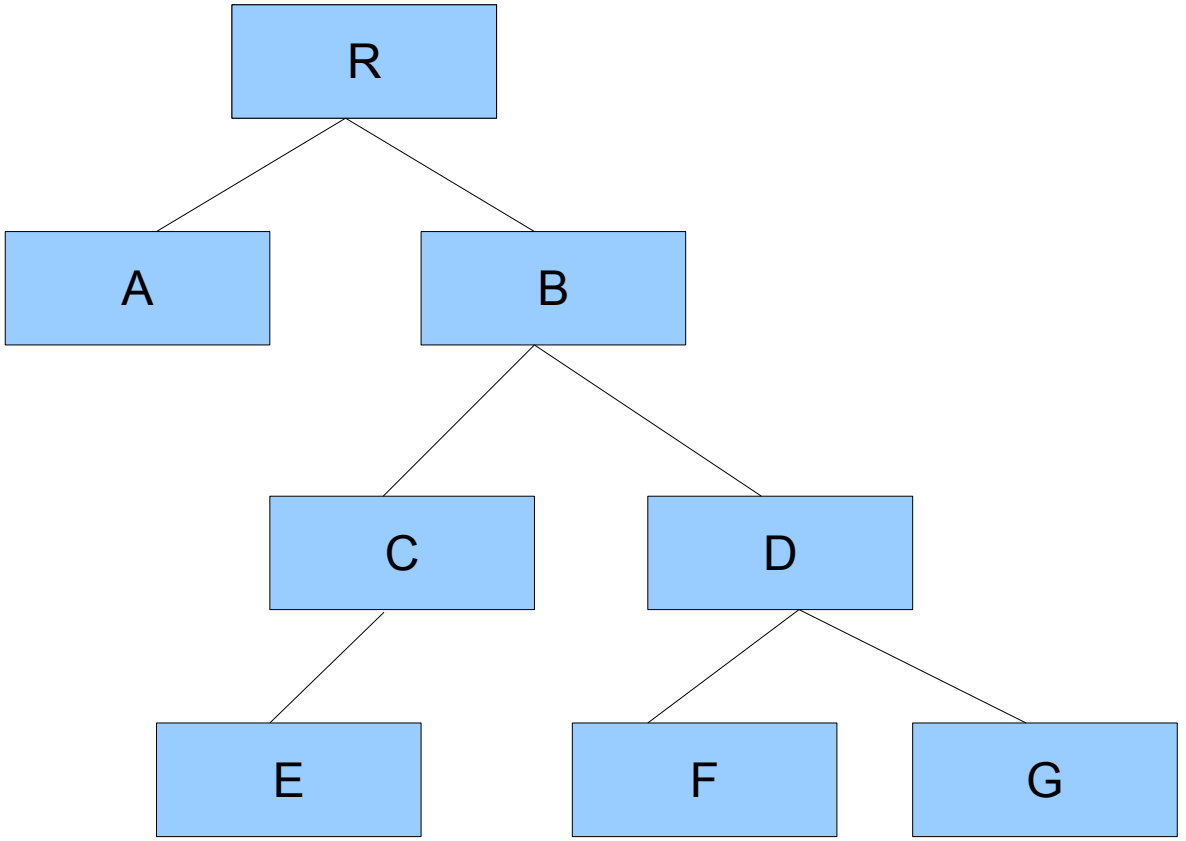
Continuous integration

...

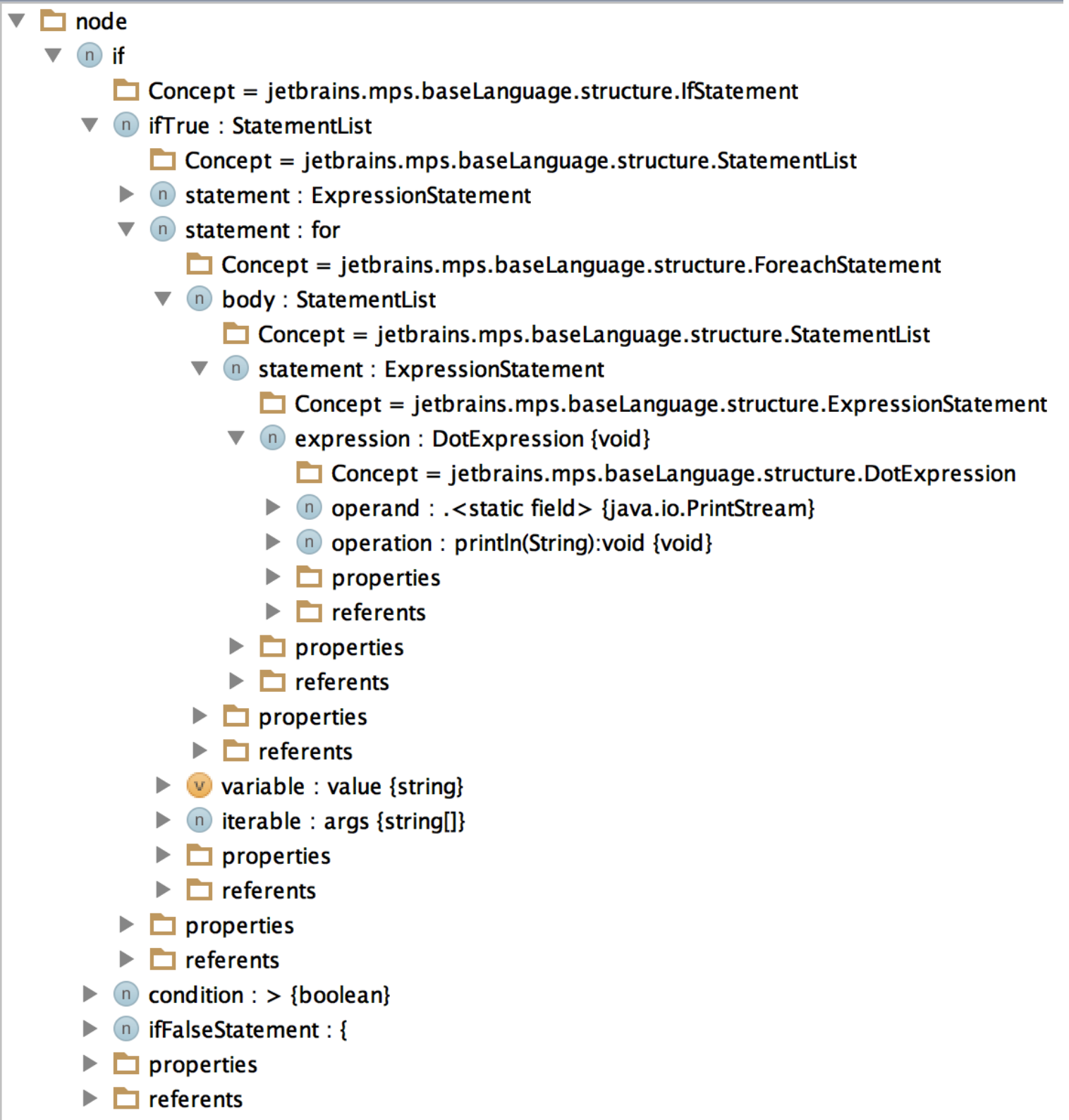
# Programming



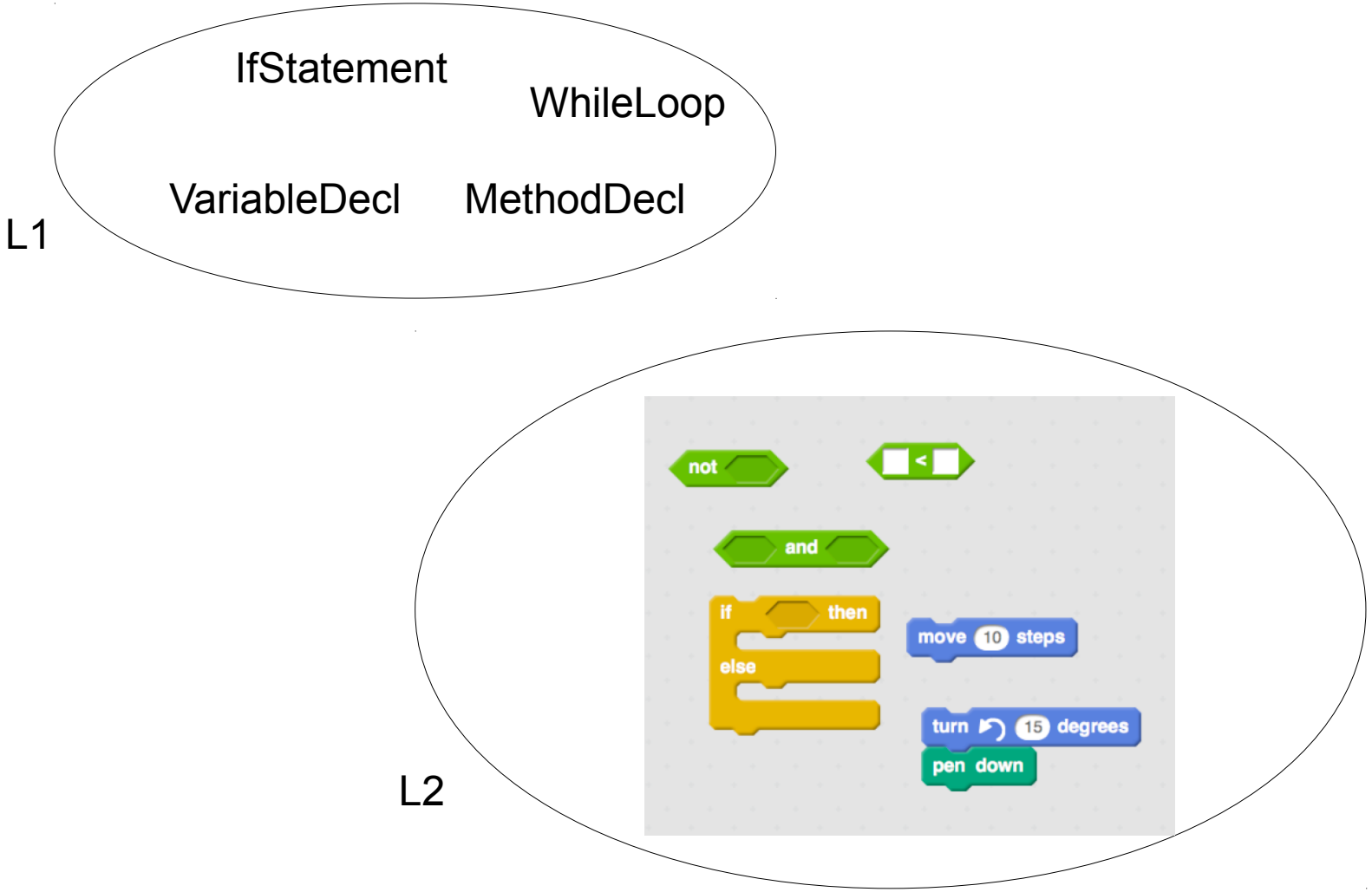
# Programs are trees



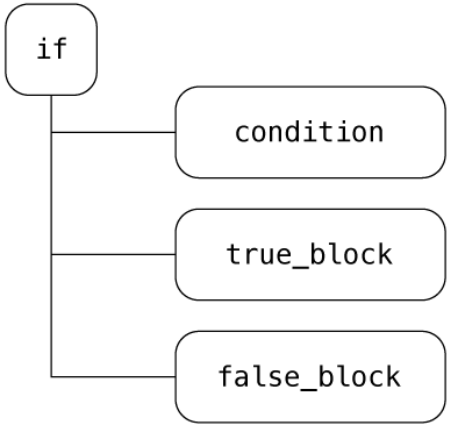

```
public class Demo {

  private static void foo() {
    System.out.println("Foo called");
  }

  public static void main(string[] args) {

    System.out.println("Application started");

    if (args.length > 0) {
      System.out.println("Supplied arguments");
      for (string value : args) {
        System.out.println("Argument: " + value);
      }
    } else {
      System.out.println("No arguments provided");
    }

    foo();
    System.out.println("Application completed");
  }

}
```


node
if
Concept = jetbrains.mps.baseLanguage.structure.IfStatement
ifTrue : StatementList
Concept = jetbrains.mps.baseLanguage.structure.StatementList
statement : ExpressionStatement
statement : for
Concept = jetbrains.mps.baseLanguage.structure.ForeachStatement
body : StatementList
Concept = jetbrains.mps.baseLanguage.structure.StatementList
statement : ExpressionStatement
Concept = jetbrains.mps.baseLanguage.structure.ExpressionStatement
expression : DotExpression {void}
Concept = jetbrains.mps.baseLanguage.structure.DotExpression
operand : .<static field> {java.io.PrintStream}
operation : println(String):void {void}
properties
referents
properties
referents
properties
referents
variable : value {string}
iterable : args {string[]}
properties
referents
properties
referents
condition : > {boolean}
ifFalseStatement : {
properties
referents

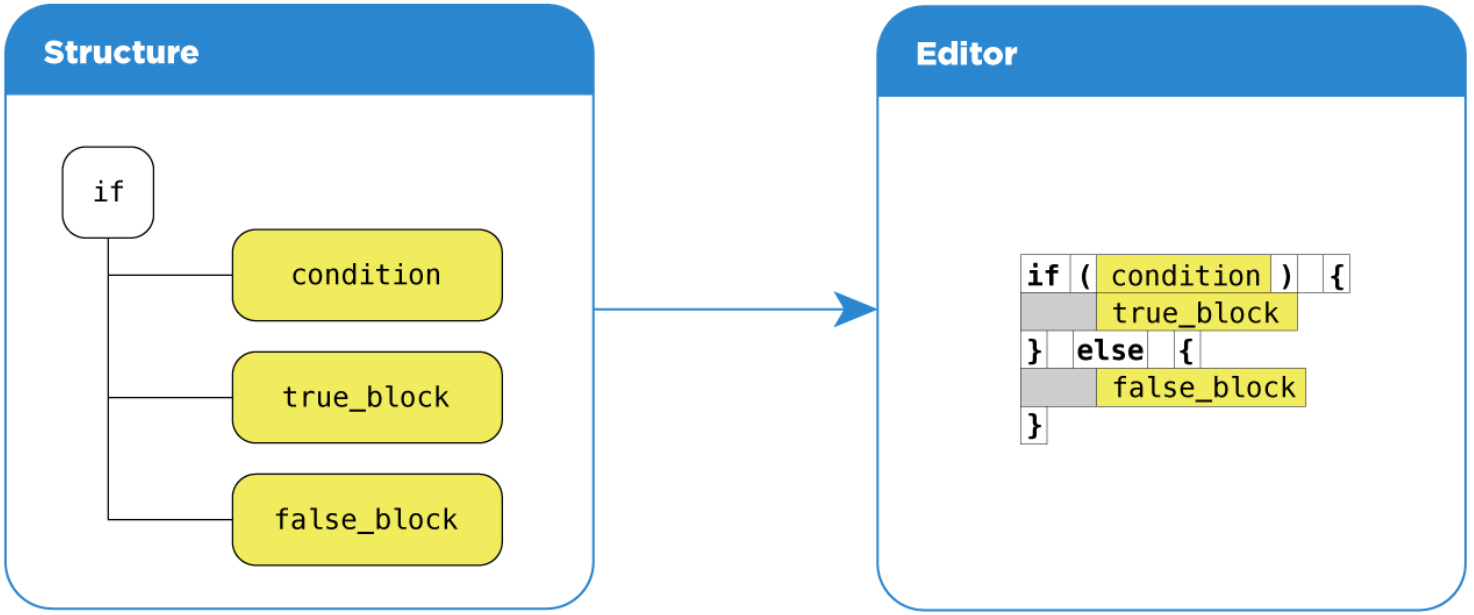
# Languages are sets of concepts



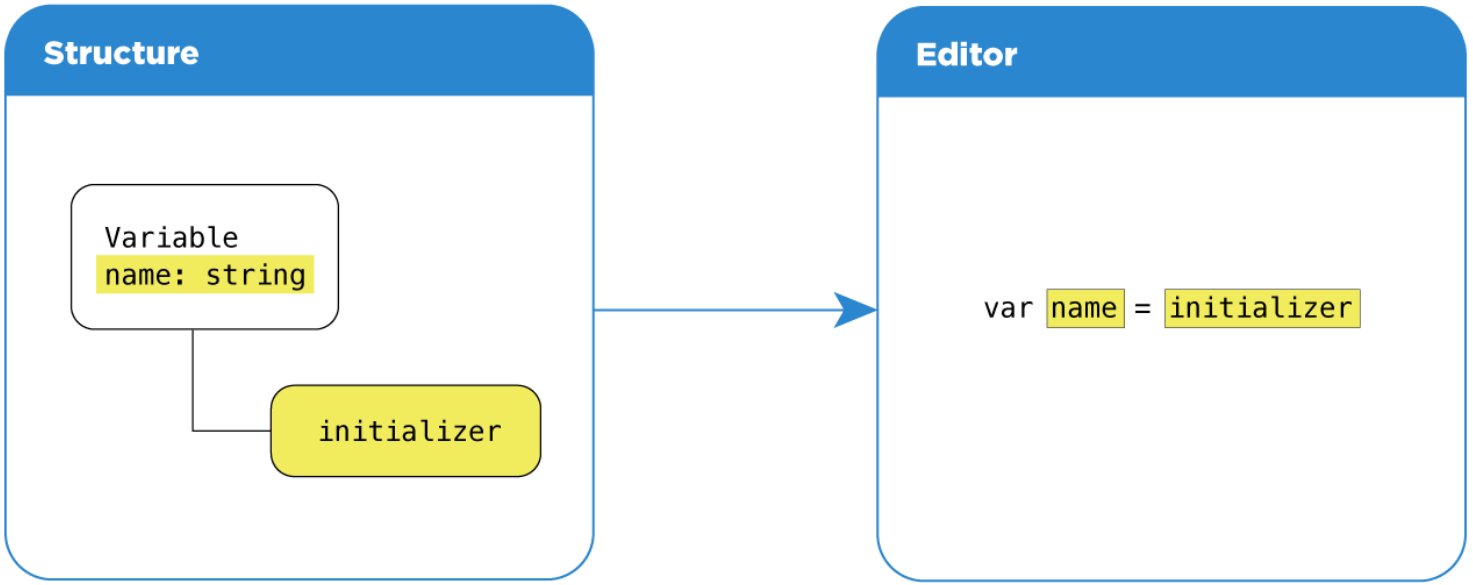
# If statement



# Abstract and concrete notation



# Abstract and concrete notation

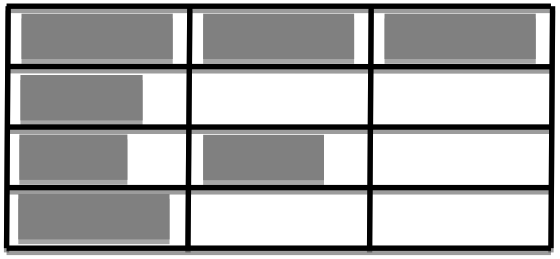


# Notations for DSLs

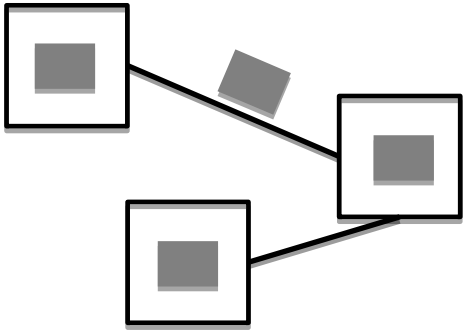
**Regular Code/Text**

**Mathematical**


**Tables**



**Graphical**



# Syntactic Flexibility

## Regular Code/Text

```
//[ A documentation comment with references
  [ to @arg(data) and @arg(dataLen)
void aSummingFunction(int8[ ] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

## Mathematical

```
double midnight2(int32 a, int32 b, int32 c) {
  return (-b + √(b² - Σ_{i=1}^{4} a * c)) / (2 * a);
} midnight2 (function)
```

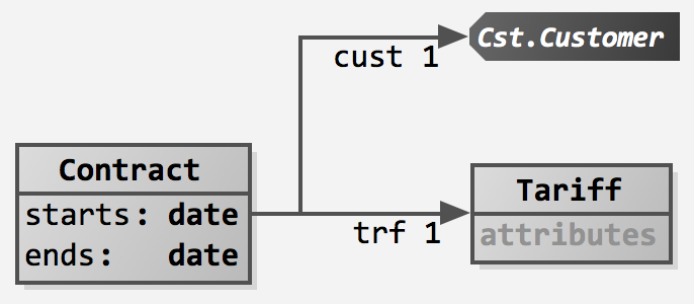
## Tables

```
int16 decide(int8 spd, int8 alt) {
  return
```

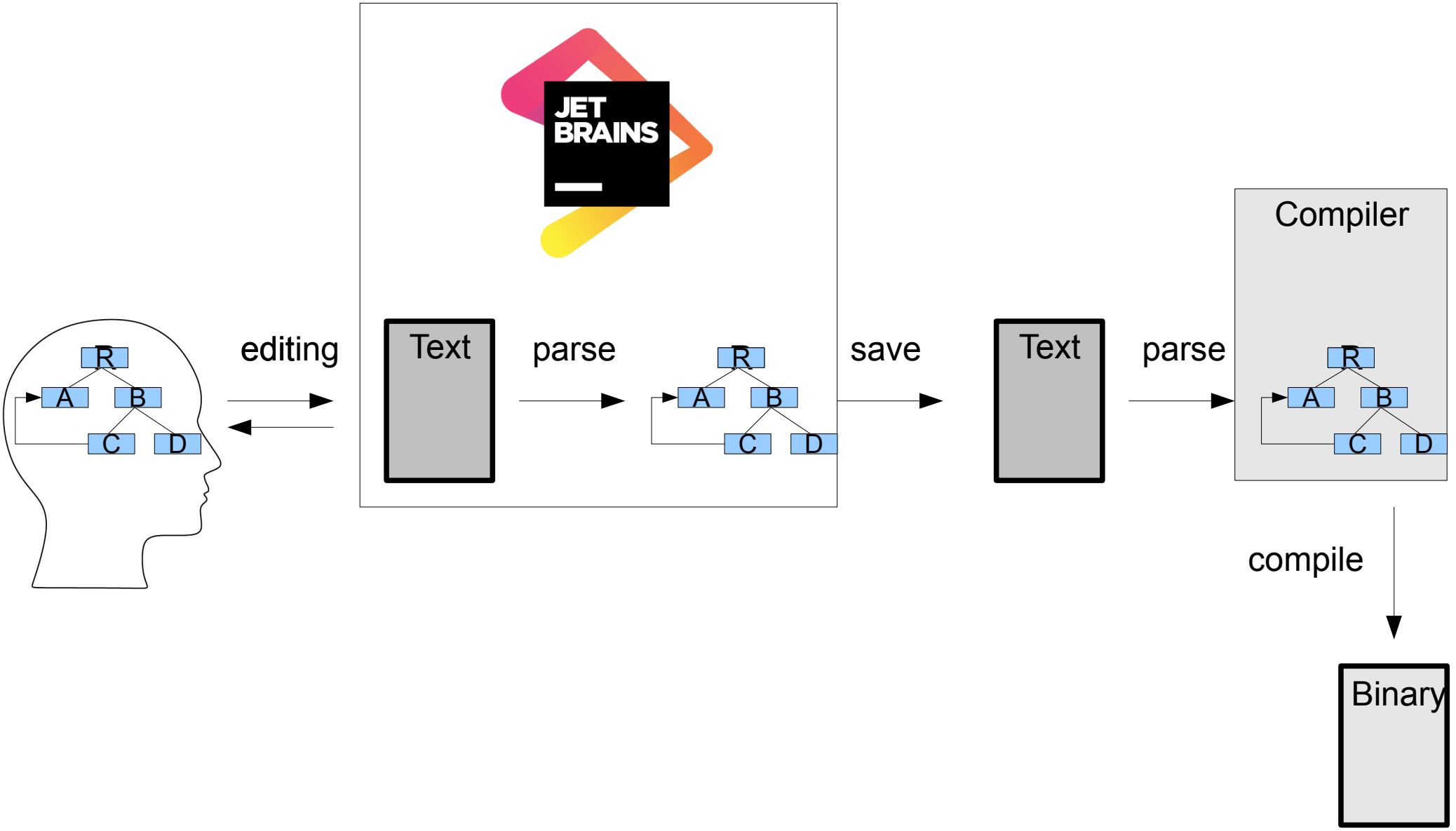
| | spd > 0 | spd > 100 |
|---|---|---|
| alt < 0 | 1 | 1 |
| alt == 0 | 10 | 20 |
| alt > 0 | 30 | 40 |
| alt > 100 | 50 | 60 |

```
} decide (function)
```

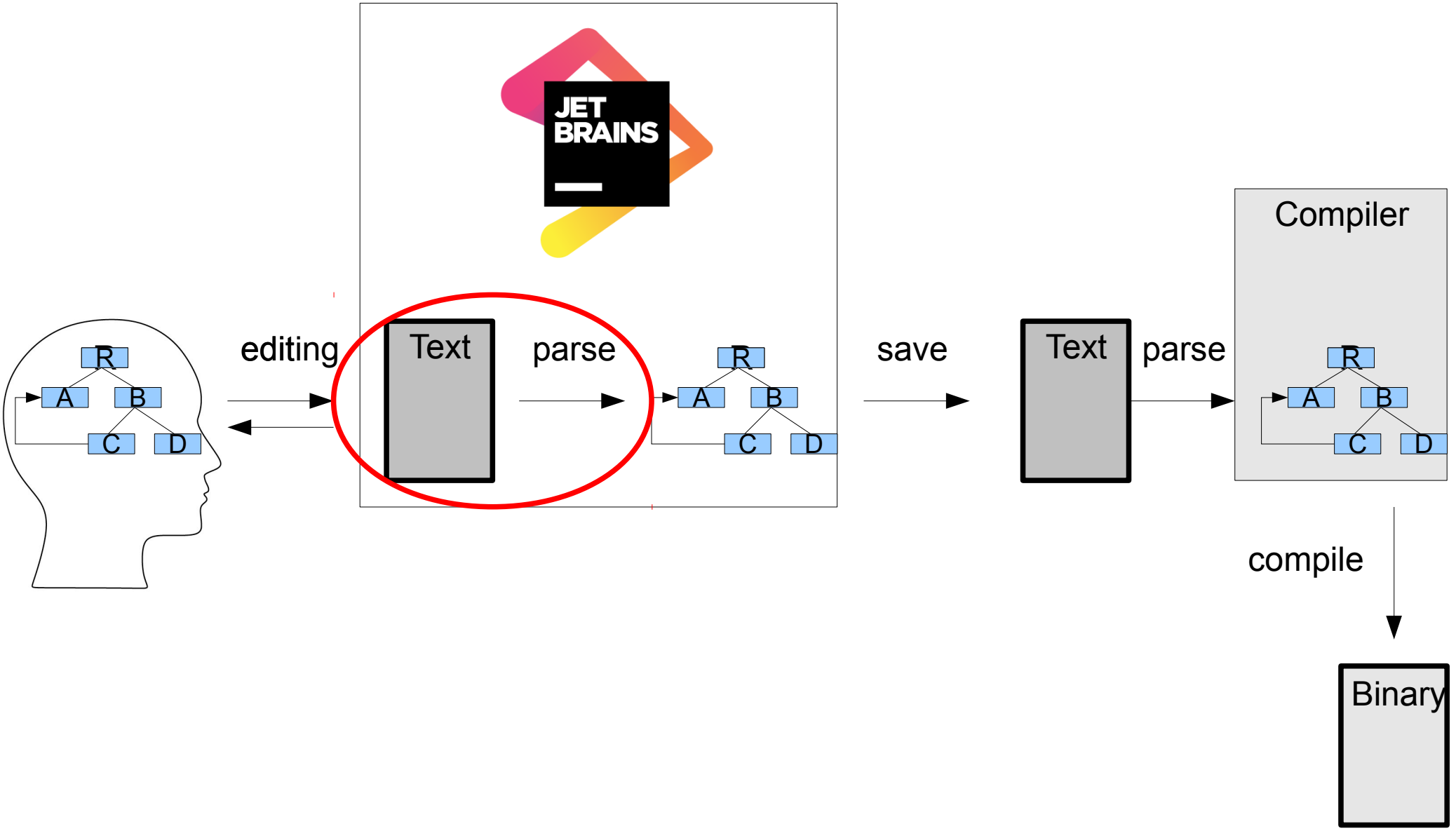
## Graphical



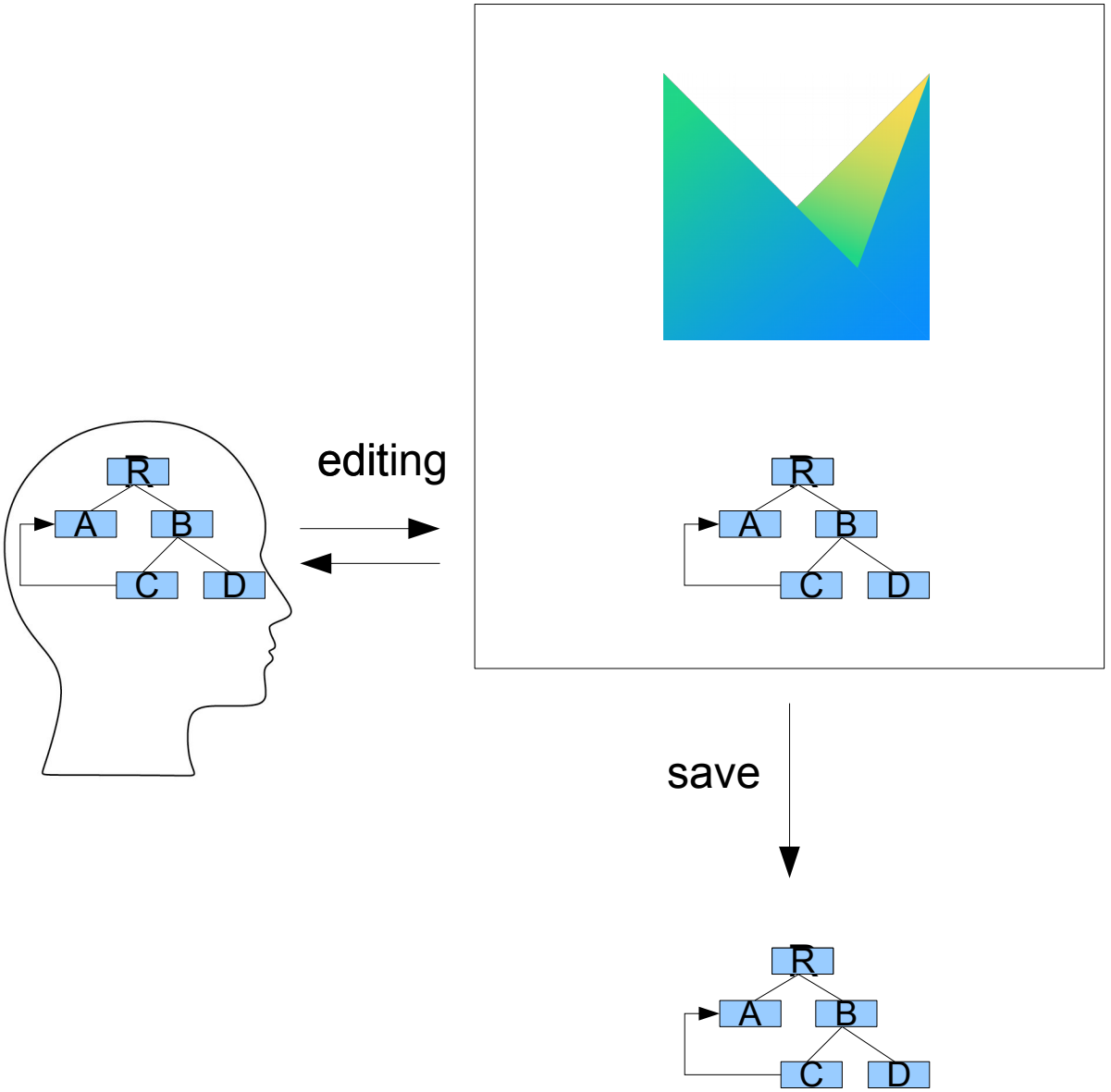
# Programming



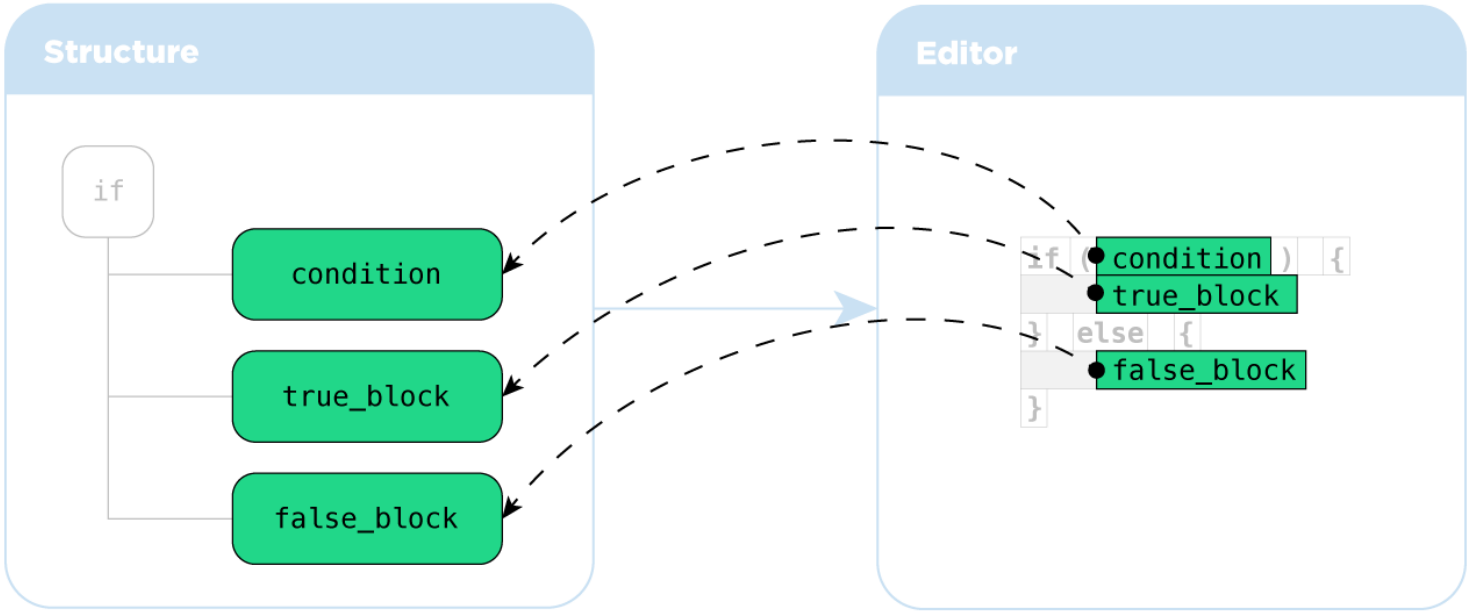
# Textual editing

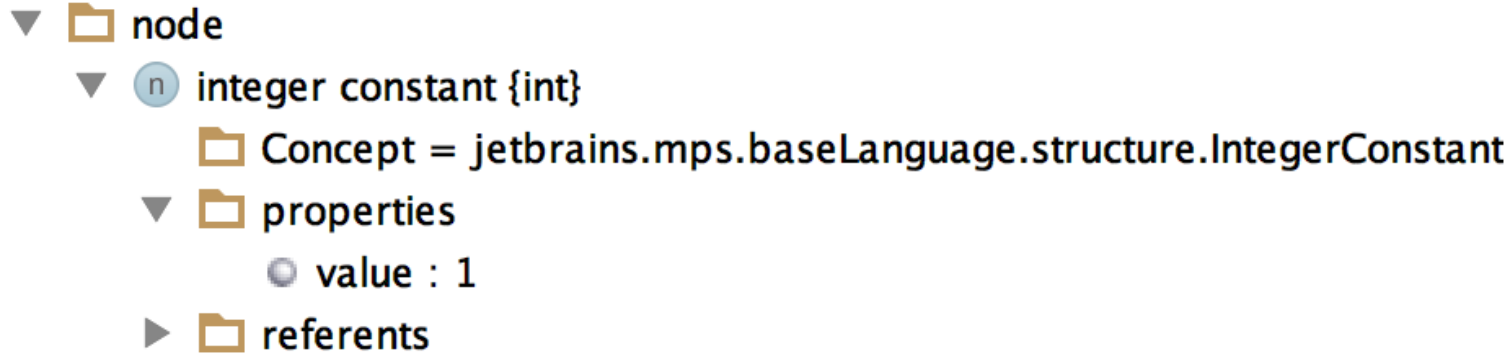


# Projectional editing

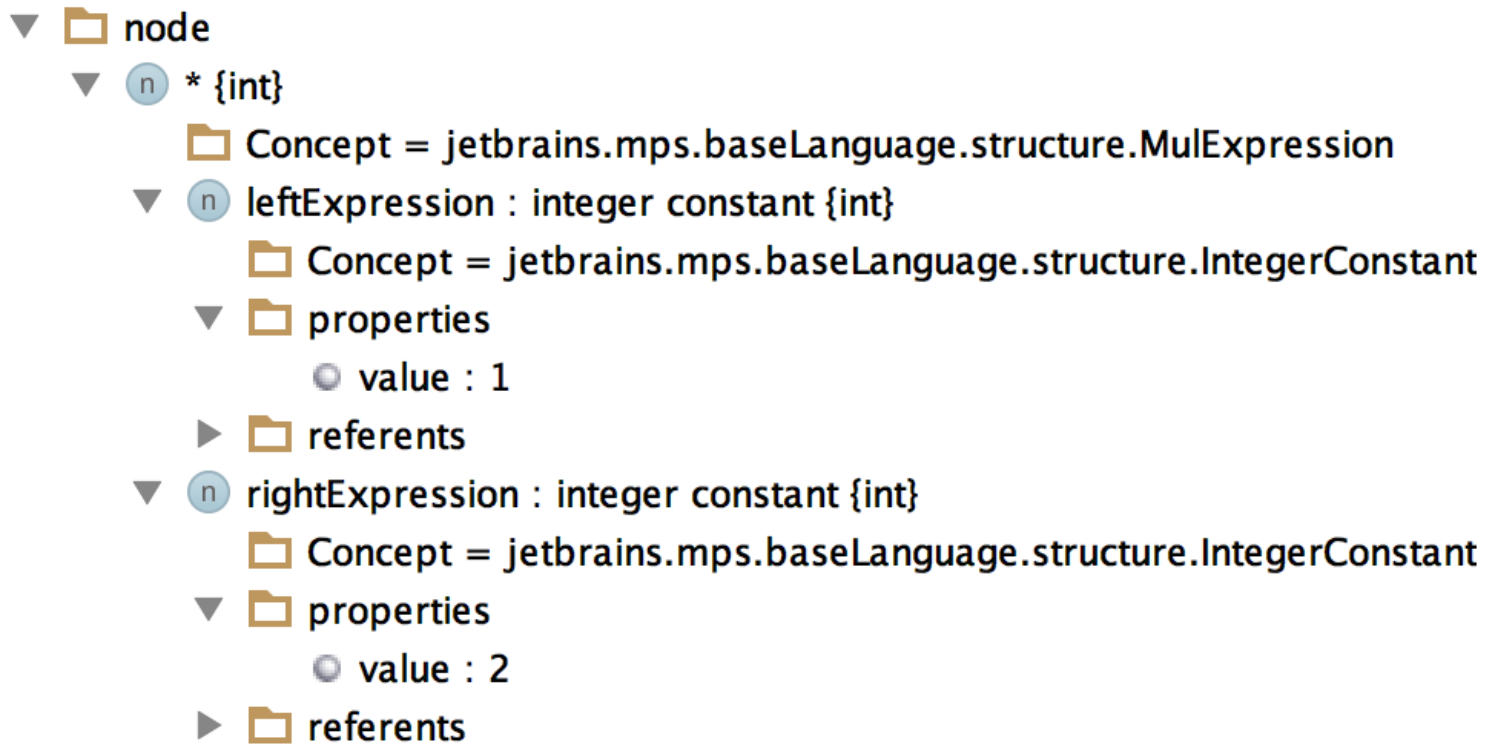


# Abstract and concrete notation

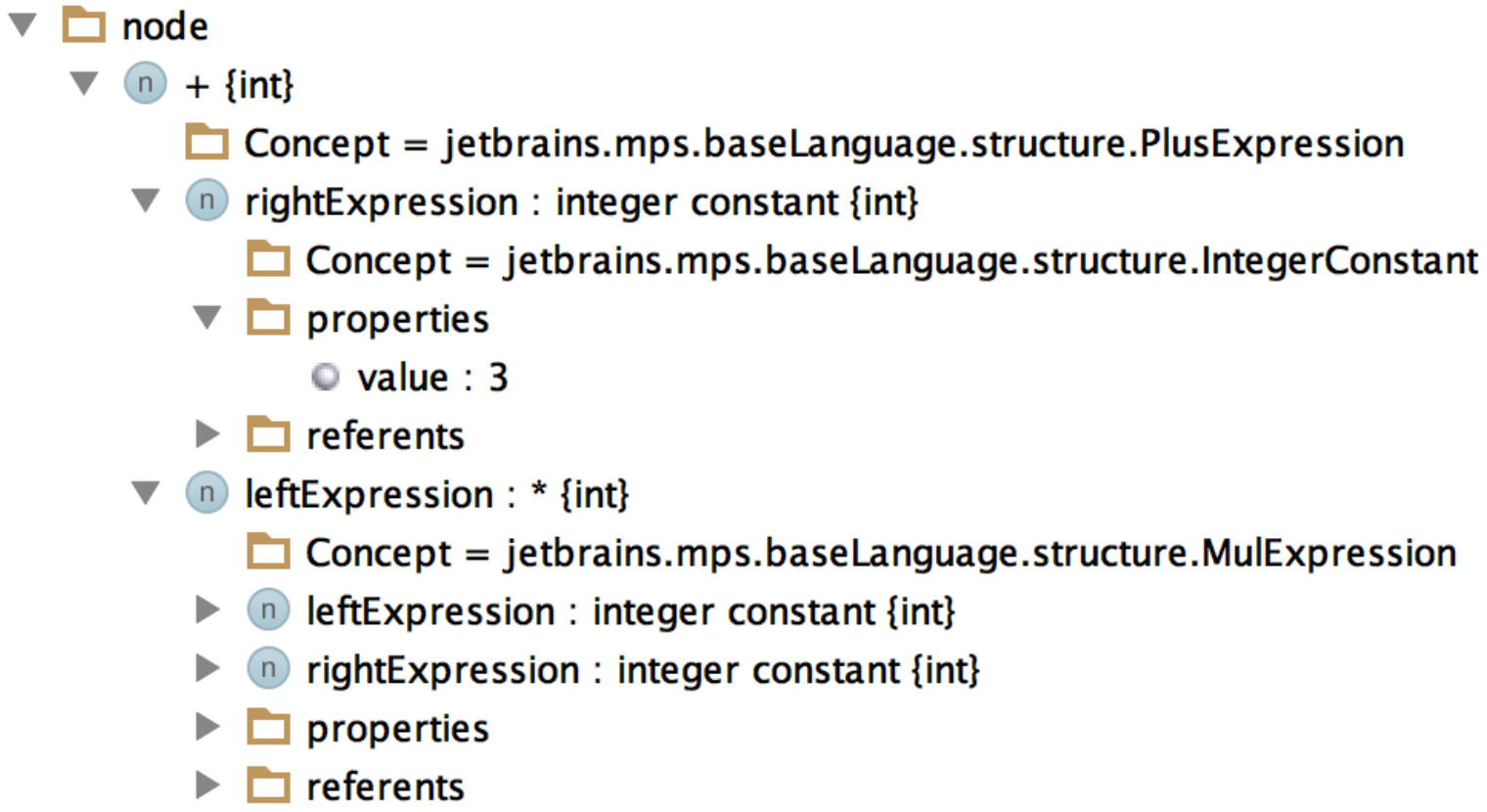



node
integer constant {int}
Concept = jetbrains.mps.baseLanguage.structure.IntegerConstant
properties
value : 1
referents

```
int a = 1;
```


node
* {int}
Concept = jetbrains.mps.baseLanguage.structure.MulExpression
leftExpression : integer constant {int}
Concept = jetbrains.mps.baseLanguage.structure.IntegerConstant
properties
value : 1
referents
rightExpression : integer constant {int}
Concept = jetbrains.mps.baseLanguage.structure.IntegerConstant
properties
value : 2
referents

```
int a = 1 * 2;
```


node
+ {int}
Concept = jetbrains.mps.baseLanguage.structure.PlusExpression
rightExpression : integer constant {int}
Concept = jetbrains.mps.baseLanguage.structure.IntegerConstant
properties
value : 3
referents
leftExpression : * {int}
Concept = jetbrains.mps.baseLanguage.structure.MulExpression
leftExpression : integer constant {int}
rightExpression : integer constant {int}
properties
referents

```
int a = 1 * 2 + 3;
```

# Rich syntaxes

```
public class Gol {
  public static void main(String[] args) {
    new Gol().run();
  }
  public void run() {
    sequence<Coordinate> generation = new arraylist<Coordinate>{(4, 5), (5, 5), (6, 5)};
    for (int i = 0; i < 5; i++) {
      System.out.println("Next generation: " + generation);
      generation = nextGeneration(generation);
    }
  }
  private sequence<Coordinate> nextGeneration(sequence<Coordinate> generation) {
    set<Coordinate> candidates = new hashset<Coordinate>;
    candidates.addAll(generation);
    generation.forEach({~it => candidates.addAll(neighbors(it)); });
    set<Coordinate> nextGeneration = new hashset<Coordinate>;
    foreach c in candidates {
      if (boolean Default: dead
```

| | generation.contains(c) | !generation.contains(c) |
|---|---|---|
| countAliveNeighbors(generation, c) < 2 | dead | dead |
| countAliveNeighbors(generation, c) == 2 | alive | dead |
| countAliveNeighbors(generation, c) == 3 | alive | alive |
| countAliveNeighbors(generation, c) > 3 | dead | dead |

```
      ) {
        nextGeneration.add(c);
      }
    }
    return nextGeneration;
  }
  private sequence<Coordinate> neighbors(Coordinate cell) {
    ((cell +
```

| | left | middle | right |
|---|---|---|---|
| upper | (-1, 1) | (0, 1) | (1, 1) |
| middle | (-1, 0) | (0, 0) | (1, 0) |
| lower | (-1, -1) | (0, -1) | (1, -1) |

```
    ) - cell).asSequence;
  }
  private int countAliveNeighbors(sequence<Coordinate> currentGeneration, Coordinate cell) {
    return neighbors(cell).intersect(currentGeneration).size;
  }

}
```

by Jens Nerche @jensnerche

# Tabular notations

```
[checked]
exported statemachine FlightAnalyzer initial = beforeFlight {
```

| | | Events | |
|---|---|---|---|
| | | next(Trackpoint* tp) | reset() |
| States | beforeFlight | [tp->alt > 0 m] -> airborne | |
| | airborne | [tp->alt == 0 m && tp->speed == 0 mps] -> crashed<br>[tp->alt == 0 m && tp->speed > 0 mps] -> landing<br>[tp->speed > 200 mps && tp->alt == 0 m] -> airborne<br>{ points += VERY_HIGH_SPEED; }<br>[tp->speed > 100 mps && tp->speed <= 200 mps && tp->alt == 0 m] -> airborne<br>{ points += HIGH_SP | [ ] -> beforeFlight |
| | landing | [tp->speed == 0 mps]<br>[tp->speed > 0 mps] -<br>{ points--; } | |
| | landed | | |
| | crashed | | |

```
}
```

**Core Data** DefaultRegions **for entity** BillingRegion

| Code | Name | Base Min Price | Max Rebate Factor |
|---|---|---|---|
| BW | Baden Württemberg | 0.20 | 0.8 |
| BY | Bayern | 0.20 | 0.8 |
| BE | Berlin | 0.15 | 0.7 |
| BB | Brandenburg | 0.10 | 0.7 |
| HB | Bremen | 0.20 | 0.7 |
| HH | Hamburg | 0.15 | 0.7 |
| HE | Hessen | 0.15 | 0.7 |
| MV | Mecklenburg-Vorpommern | 0.10 | 0.7 |
| NI | Niedersachsen | 0.15 | 0.7 |
| NW | Nordrhein-Westfalen | 0.15 | 0.7 |
| RP | Rheinland-Pfalz | 0.15 | 0.7 |
| SL | Saarland | 0.15 | 0.7 |
| SN | Sachsen | 0.10 | 0.7 |
| ST | Sachsen-Anhalt | 0.10 | 0.7 |
| SH | Schleswig-Holstein | 0.15 | 0.7 |
| TH | Thüringen | 0.10 | 0.7 |

# Symbols

```
int32 sumUpIntArray(int32[ ] arr, int32 size) {
  return Σ_{i = 0}^{size} arr[i] ;
} sumUpIntArray (function)
```

```
int32 averageIntArray(int32[ ] arr, int32 size) {
  return (Σ_{i = 0}^{size} arr[i]) / size ;
} averageIntArray (function)
```

```
double midnight1(int32 a, int32 b, int32 c) {
  return (-b + √(b^2 - 4 * a * c)) / (2 * a) ;
} midnight1 (function)
```

```
double midnight2(int32 a, int32 b, int32 c) {
  return (-b + √(b^2 - Σ_{i = 1}^{4} a * c)) / (2 * a) ;
} midnight2 (function)
```

```
double sumOfProductsOfLogs(int32[ ] arr, int32 size) {
  return Σ_{k = 0}^{size} (Π_{i = 0}^{k} log_2 arr[i]) / 2 ;
} sumOfProductsOfLogs (function)
```

# Positional notations

**Rule Set Type** DemoRuleSetType

**Business objects**

person : **Person**

**Variables:**

PRMI : **int**
FR : **int**
NN : **int**
TT : **int**
J : **int**
A3 : **int**
G3 : **int**
ANUI : **int**
X : **int**

**Parent**

<no parent>

**Libraries**

Standard
Extra

**Rule Set Type** DemoRuleSetType

**Business objects**

<no business objects>

**Variables:**

<no variables>

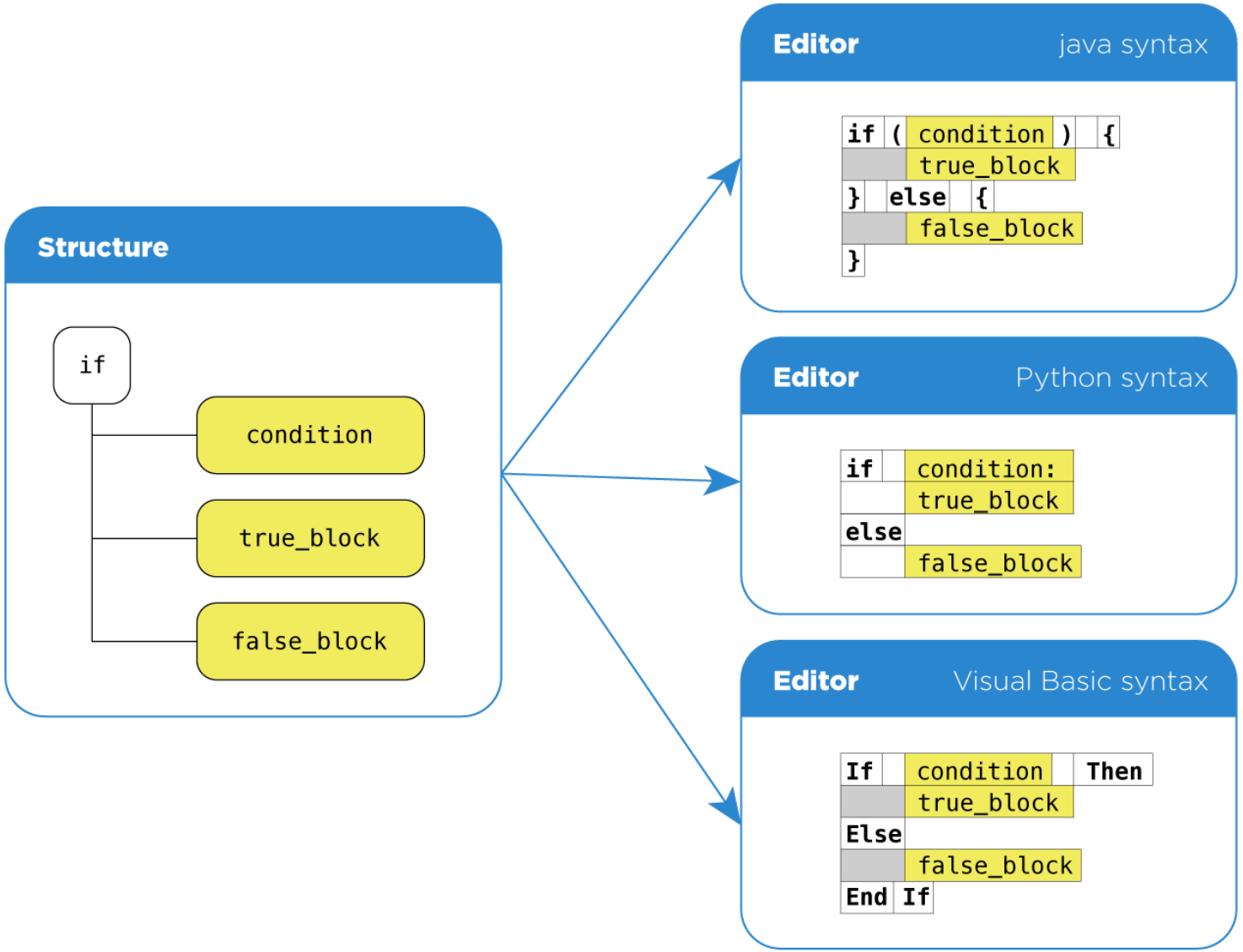
**Parent**

<no parent>

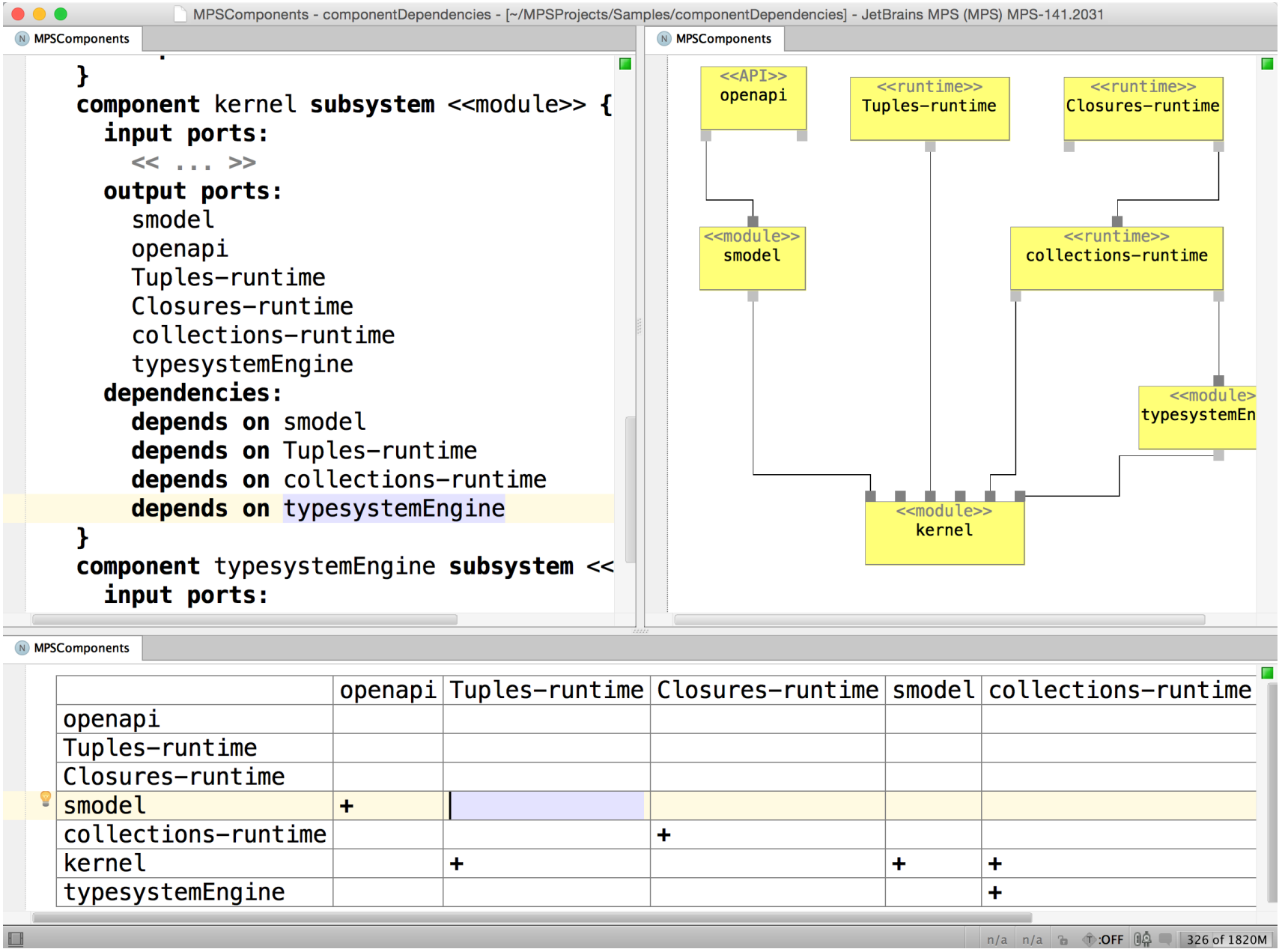
**Libraries**

<no libraries>

# Multiple switchable notations



# Multiple switchable notations



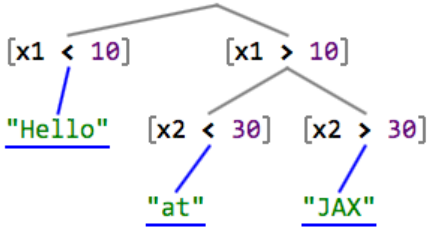
# Combine languages

```
variables:
  int x1             = 10 * (1 + 2)                    ==> 30
  int x2             = 20                              ==> 20
  boolean b1         = true || !false                  ==> true
  int b2             = if [ b1 then 12 else 13 ]       ==> 12
  list<int> intList  = list(1, 2, 3)                   ==> [1, 2, 3]
  int three          = intList.last                    ==> 3
  list<int> t2       = intList.where|it > 2|           ==> [3]
  boolean allEl      = intList.all|it > 0|             ==> true
  int surprise2      = doWithTwoInts(:add, 1, 3)       ==> 4
  [int, int] tuple   = [1, 2]                          ==> [1, 2]
  int one            = tuple[0]                        ==> 1
  int c1             = alt [x1 < 0 && x2 > 1  => 2]    ==> 0
                           [x1 > 0 && x2 == 1 => 1]
  int c2             =                                 ==> 9
```

| | x1 < 0 | x1 == 0 | x1 > 0 |
|---|---|---|---|
| x2 < 0 | 1 | 2 | 3 |
| x2 == 0 | 4 | 5 | 6 |
| x2 > 0 | 7 | 8 | 9 |

```
  int complicated    = {
                         val t1 = 10 + 20
                         val t2 = t1 + 30
                         t2
                       }
```

```
string res         =                                  ==> JAX
```




```
functions:
  fun add(int a, int b)                                    : int                = a + b
  fun doWithTwoInts((int, int => int) fun, int a, int b)   : int                = fun.exec(a, b)
  fun anotherFun(option<int> i)                            : int                = with some i as x => x + 1 none 20
  fun giveMeAnInt()                                        : int                = anotherFun(some(10))
  fun getStreets(Person p)                                 : collection<string> = p.workedAt.offices.street
```

# Parsing is the bottleneck

… of language expressiveness

- Limits the **possible** syntaxes
- Allows only **one** editable code visualization
- Complicates **combining** languages

The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.

*Edsger Dijkstra*