Build your own DSL with Rascal

Tijs van der Storm

university of groningen

CWI

Rascal = Functional Metaprogramming language

Rascal = Functional Metaprogramming language

- Immutable values
- Higher order functions
- Static safety, with local type inference

Rascal = Functional Metaprogramming

???

Metaprogramming

From Wikipedia, the free encyclopedia

Metaprogramming is a programming technique in which computer programs have the ability to treat other programs as their data. It means that a program can be designed to read, generate, analyze or transform other programs […]

Rascal = Functional Metaprogramming

- “Code as data”
- Program that generates/analyzes other programs
- Syntax definitions and parsing
- Pattern matching and rewriting mechanisms
- Visiting / traversal of Tree structure

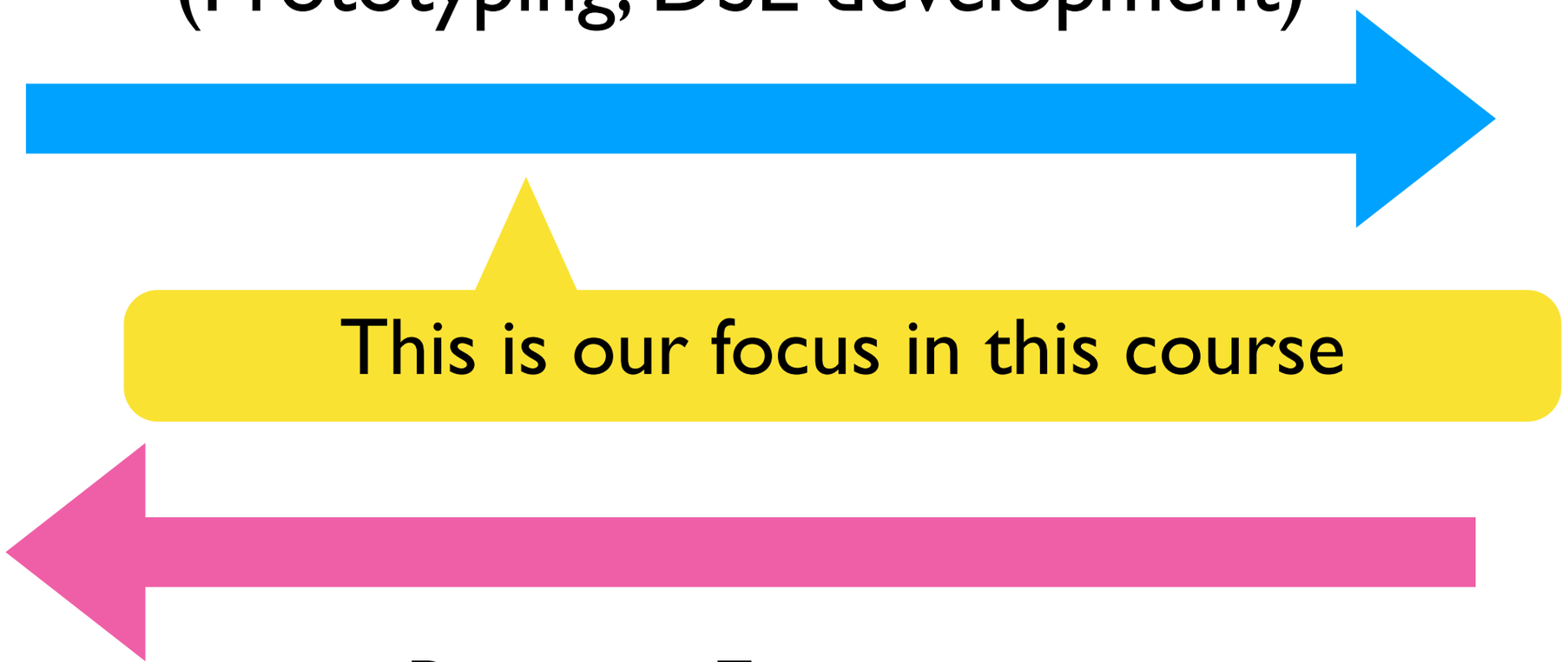
Rascal can be used for…

Forward Engineering
(Prototyping, DSL development)

Reverse Engineering
(Analysis, detectors, renovations)

Rascal can be used for…

Forward Engineering
(Prototyping, DSL development)



Reverse Engineering
(Analysis, detectors, renovations)

Why learn *yet* another new language?


Java
Grep
ANTLR
awk

Basic concepts

Functional immutability with an imperative syntax

- All data is immutable

- Can write code that looks like ‘mutating variables’

Functional immutability with an imperative syntax

- **All data is immutable**

- Can write code that looks like ‘mutating variables’

Common Data Types

- e.g. Sets, Lists, Maps, Tuples and Relations
  - `{1,2,3}`
  - `[1,2,3]`
  - `(1:2, 3:4)`
  - `{<“a”, “b”>, <“b”, “c”>}`
- Can all be used in comprehensions
  - `{ i*i | int i ← [1..10] } etc.`

Source Locations

- Provide a uniform way to represent files on local or remote storage

- Can have different schemes
  - `file:///`
  - `project://`
  - `http://`
  - etc …

- Can contain text location markers

- `|project://rascal-dsl-crashcourse/examples/errors.myql|(0,471,<1,0>,<21,1>)`

Pattern Matching

- Determines whether pattern matches a given value
- One of Rascal's most powerful features
- Can bind matches to local variables
- Can be used in many places
- May result in multiple matches so employs local backtracking

Different types of matching

type-based matching

```
int x := 3;
```

structural matching

```
event(x, y) := event("a", "b");
```

anti-matching

```
event("c", "d") !:= event("a", "b");
```

list matching

```
[*x, 1, *y] := [5, 6, 1, 1, 1, 3, 4];
```

set matching

```
{1, *x} := {4, 5, 6, 1, 2, 3};
```

deep matching

```
/transition(e, "idle") := ast;
/state(x, _, /transition(_, x)) := ast;
```

element matching

```
3 ← {1,2,3}
int x ← {1,2,3}
```

regular expressions

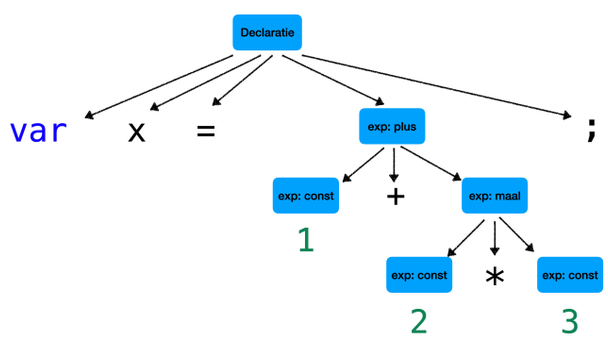
```
/[A-Za-z]*/ := “09090aap noot mies”
```

Tools for Language Engineering

Basic language pipe line

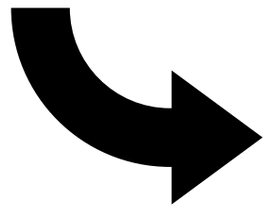
```
var x = 1 + 2 * 3;
```

Parse



Check

Execute



Result

Describes all possible strings which can be produced

```
start syntax Program = "begin" Stat* "end";

syntax Statement
  = "i                  " Stat* "else" Stat* "fi"
  | Id
  | "while" Expression "do" Stat* "od"
  ;

syntax Expression
  = Id
  | "(" Expression ")"
  | left Expression "*" Expression
                    on "+" Expression

lex                                  -9\-]*;

layout Whitespace = [\ \t\n\r]*;
```

Nonterminal

Terminal

Layout characters

Tips and Tricks

Common errors
Undeclared variable

- Forgetting to import a module, i.e.:

- Function is declared private

```
rascal>l = [1,2,3];
list[int]: [1,2,3]
rascal>size(l);
|prompt:///|(0,4,<1,0>,<1,4>): Undeclared variable: size
Advice: |http://tutor.rascal-mpl.org/Errors/Static/UndeclVariable.html|
```

- Solution for above example: `import List;`

Common errors
CallFailed

- Calling a function with the wrong arguments

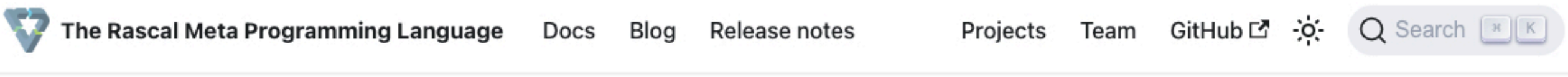
```
void someFunc(str a) {
  println(a);
}
```

```
rascal>someFunc("a");
a
ok
rascal>someFunc(2);
|prompt:///|(9,1,<1,9>,<1,10>): CallFailed(
  |prompt:///|(9,1,<1,9>,<1,10>),
  [2])
        at $root$(|prompt:///|(0,12,<1,0>,<1,12>))
```

Looking for how you can do stuff in Rascal?

Browse and search the documentation

- https://www.rascal-mpl.org/docs/GettingStarted/



The Rascal cheatsheet

Rascal Cheat Sheet

http://www.rascal-mpl.org
https://github.com/usethesource/rascal

Modules

```
module Example

import ParseTree;         // import
extend lang::std::Layout; // "inherit"
```

Declarations

```
// Algebraic data types (ADT)
data Exp
  = var(str x)         // unary constructor
  | add(Exp l, Exp r); // binary constructor

data Person        // keyword parameter
  = person(int id, bool married=false);

alias Age = int; // type alias

anno loc Exp@location; // annotation

private real PI = 3.14; // variables

// Functions:  signatures are lists of patterns
// May have keyword parameters.
void f(int x) { println(x); }      // block style
int inc(int x) = x+1;              // rewrite style
int inc0(int x) = x+1 when x == 0; // side condition
default int inc0(int x) = x;       // otherwise

// Test functions (invoke from console with :test)
test bool add() = 1+2 == 3;

// randomized test function
test bool comm(int x, int y) = x+y == y+x;

// Foreign function interface to Java
@javaClass{name.of.javaClass.with.Method}
java int method();
```

```
// Context-free grammars
start syntax Prog       // start symbol
  = prog:  Exp* exps    // production
  | stats:  {Stat ";"}* // separated list
  | stats:  {Stat ";"}+ // one-or-more sep.  list
  | "private"?  Func;   // optional

syntax Exp
  = var:  Id
  | left mul:  Exp l "*" Exp r     // or right, assoc
  | left div:  Exp!div "/" Exp!div // reject
  > left add:  Exp l "+" Exp r     // ">" = priority
  | bracket "(" Exp ")";

lexical Comment
  = ^"#" ![\n]* $; // begin/end markers

lexical Id
  = ([a-zA-Z] !<<         // look behind restriction
    [a-zA-Z][a-zA-Z0-9_]* // character classes
    !>> [a-zA-Z0-9_])     // lookahead restriction
    \ Reserved;           // subtract keywords

layout Layout // for whitespace/comments
  = [\ \t\n\r]*;

keyword Reserved   // keyword class
  = "if" | "else"; // finite langs
```

Statements

```
// Standard control-flow
if (E) S;
if (E) S; else S;
while (E) S;
do S; while(E);
continue; break;
return; return E;

// Loop over all bindings produce by patterns
for (i <- [0..10]) S; // Loop 10 times

fail;     // control backtracking
append E; // add to loop result list
```

```
// Pattern-based switch-case
switch (E) {
  case P: S;   // do something
  case P => E  // rewrite it
  default:  S; // otherwise
}

// Traversal with visit; like switch, but matches
// at arbitrary depth of value
visit (E) {
  case P: S;  // do something
  case P => E // rewrite something
  case P => E when E
}
insert E; // rewrite subject as statement

// Strategies:  bottom-up, innermost, outermost,
// top-down-break, bottom-up-break
top-down visit (E) {}

try S;      // pattern-based try-catch
catch P: S; // match to catch
finally S;

throw E; // throw values

// Fix-point equation solving;
// iterates until all params are stable
solve (out,ins) {
  out[b] = ( {} | it + ins[s] | s <- succ[b] );
  ins[b] = (out[b] - kill[b]) + gen[b];
};

x = 1;               // assignment
nums[0] = 1;         // subscript assignment
nums[1,3..10] = 2;   // sliced (see below)
p.age = 31;          // field assignment
ast@location = l;    // annotation update
<p, a> = <"ed", 30>; // destructuring

// A op=E  == A = A op E
A += E; A -= E; A *= E;
A /= E; A &= E;
```

https://github.com/cwi-swat/rascal-cheat-sheet/raw/master/sheet.pdf

Questionnaire Language

```
form taxOfficeExample {
  "Did you buy a house in 2010?"
    hasBoughtHouse: boolean

  "Did you enter a loan?"
    hasMaintLoan: boolean

  "Did you sell a house in 2010?"
    hasSoldHouse: boolean

  if (hasSoldHouse) {
    "What was the selling price?"
      sellingPrice: integer
    "Private debts for the sold house:"
      privateDebt: integer
    "Value residue:"
      valueResidue: integer =
        sellingPrice - privateDebt

  }
}
```

Aangifte inkomstenbelasting 2010 – Persoonlijke gegevens

- ✓ Persoonlijke gegevens: Bla
- ✓ Persoonlijke gegevens: Blasa
- ✓ Box 1: werk en woning
- Box 1: andere inkomsten
- Box 1: uitgaven lijfrenten e.d.
- Box 2: aanmerkelijk belang
- Box 3: sparen en beleggen
- Aftrekposten
- Vrijstellingen en verminderingen
- Bijzondere situaties
- Te verrekenen bedragen
- Heffingskortingen: Bla
- Overzicht: Bla
- Voorlopige aanslag 2011
- Naar ondertekenen met DigiD

IB 602E - 2Z01FOL2A

Persoonlijke gegevens

Naam: Bla

Telefoonnummer: 323

Burgerservicenummer/sofinummer: 1430.95.067

? Geboortedatum: 11-02-1979

? Nummer belastingconsulent:

? Hebt u van ons bericht ontvangen om aangifte te doen? ○ Ja ● Nee

Wilt u een rekeningnummer opgeven of wijzigen? ○ Ja ● Nee

? Uw persoonlijke situatie in 2010: Een deel van 2010 getrou...

? Periode dat u getrouwd was in 2010: 01-02 03-05

? Woonde u voor of na deze periode samen met uw echtgenoot? ○ Ja ● Nee

? Willen u en uw echtgenoot heel 2010 als fiscale partners worden beschouwd? ○ Ja ● Nee

? Woonde u buiten de periode dat u getrouwd was nog met iemand anders samen? Bijvoorbeeld met uw kind van 27 jaar of ouder? ○ Ja ● Nee

Akkoord

Stoppen | Instellingen | Rekenmachine | Help | Printen | Open bestand

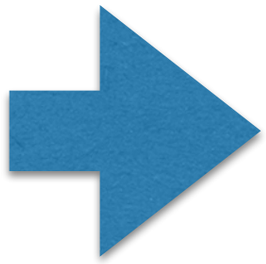
```
form taxOfficeExample {
  "Did you sell a house in 2010?"
    hasSoldHouse: boolean

  "Did you buy a house in 2010?"
    hasBoughtHouse: boolean

  "Did you enter a loan?"
    hasMaintLoan: boolean

  if (hasSoldHouse) {
    "What was the selling price?"
      sellingPrice: integer
    "Private debts for the sold house:"
      privateDebt: integer
    "Value residue:"
      valueResidue: integer =
        sellingPrice - privateDebt

  }
}
```



Did you sell a house in 2010?

Yes

Did you buy a house in 2010?

Choose an answer

Did you enter a loan?

Choose an answer

What was the selling price?

100

Private debts for the sold house:

200

Value residue:

-100.00

Submit taxOfficeExample

Tutorial (1 hour)

Open `tutorial/Series1.rsc`, click on `Run in new Rascal terminal` (top of editor). Fill in the blanks of the functions, and execute in the terminal. If you have time left, move on to `Series2.rsc`. Documentation can be found here. See also the cheat-sheet in the `slides` folder.

QL Exercises (3 hours total)

Each task corresponds to a Rascal module, see each module for an explanation:

- `Syntax.rsc`: syntax definition (grammars and parsing)
- `Check.rsc`: type checking
- `Eval.rsc`: semantics
- `App.rsc`: execution (run QL as a Salix web app)

Optional (if there's time left): extend the language with a `date` question type and revisit all the modules to adapt type checking, interpretation, and execution to support it.

https://github.com/cwi-swat/rascal-dsl-crashcourse

- 10:00-11:00 Intro lecture + tutorial (warm-up)
- 11:00-11:15 *break*
- 11:15-12:15 syntax
- 12:15-13:30 *lunch*
- 13:30-13:45 recap
- 13:45-14:45 type checking
- 14:45-15:00 *break*
- 15:00-16:00 evaluation and execution
- 16:00-16:30 wrap up