Rascal: Functional programming for source code analysis and transformation

Tijs van der Storm
storm@cwi.nl / @tvdstorm

CWI
Centrum Wiskunde & Informatica

University of Amsterdam

About me

- Researcher at CWI across the street
- Teacher at here at UvA
    - Master Software Engineering
- Interests:
    - DSLs, MDE, Meta-programming, PL
- Co-designer of Rascal (w/ Vinju, Klint)

Meta Software

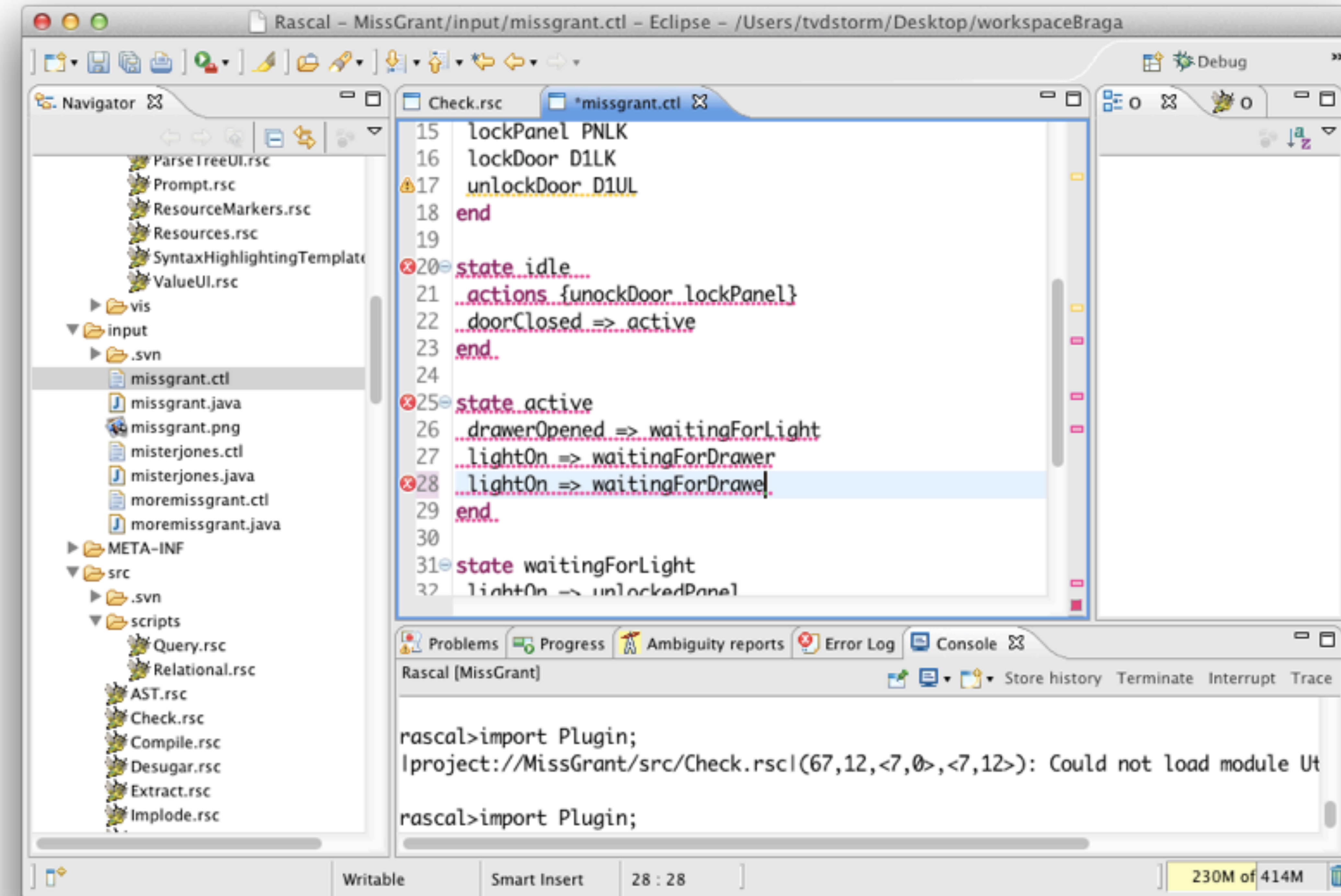
Analysis

- Dead code detection
- Slicing/Dependence
- Metrics
- Reverse engineering
- Verification
- Architecture recovery
- Code-to-model
- ...

Transformation

- Goto elimination
- Dialect transformation
- Aspect weaving
- DSL compilers
- API migration
- Model-to-code
- Model-to-model
- ...

Example application: DSLs



Rascal

http://www.rascal-mpl.org

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

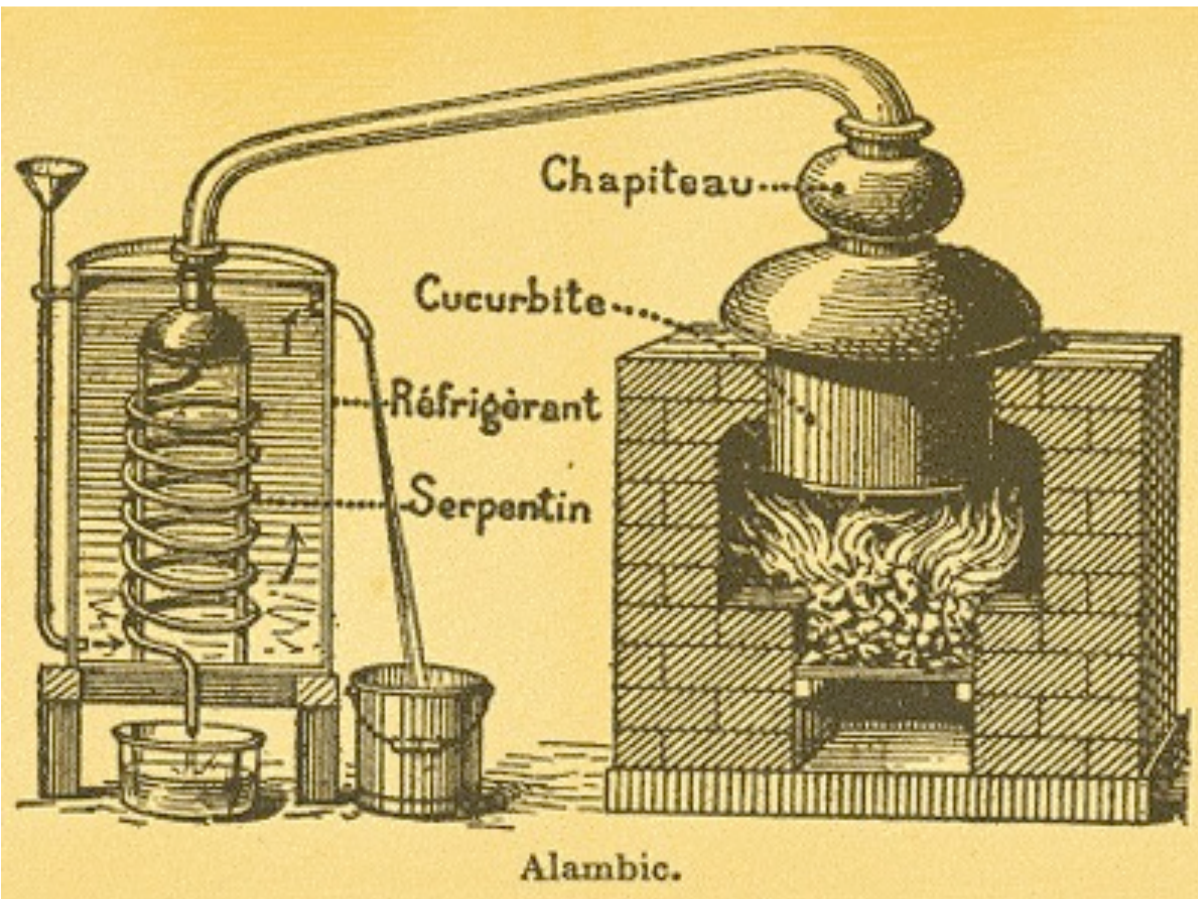
Well-known FP stuff

- Higher-order functions
- Algebraic data types
- Immutable data
- Pattern-matching (but...)
- Comprehensions (but...)

More notable

- "Java-inspired" syntax (Blasphemy! ;-)
- Built-in context free grammars & parsing
- Concrete syntax patterns and values
- Traversal primitives
- Relational calculus
- IDE hooks

Extraction and analysis



Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
   controller:
      Events events
      ResetEvents? resets
      Commands? commands
      State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
   controller:
      Events events
      ResetEvents? resets
      Commands? commands
      State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
   controller:
      Events events
      ResetEvents? resets
      Commands? commands
      State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

start symbol

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
   controller:
      Events events
      ResetEvents? resets
      Commands? commands
      State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

start symbol

production label

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
   controller:
      Events events
      ResetEvents? resets
      Commands? commands
      State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

start symbol

production label

subelement labels

Lexical syntax

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

keyword reservation

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

keyword reservation

keyword class

Parse trees

```
follow(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)]))})]))),[char(68),appl(regular(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))),[char(50),char(79),char(80)])[@loc=|project://MissGrant/input/missgrant.ctl|
(39,3,<3,15>,<3,18>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(38,4,<3,14>,<3,18>)]])[@loc=|project://MissGrant/input/missgrant.ctl|
(38,4,<3,14>,<3,18>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(25,17,<3,1>,<3,18>)],appl(prod(layouts("Standard"),[conditional(\iter-
star(sort("WhitespaceOrComment")),{\not-follow(\char-class([range(9,10),range(12,13),range(32,32)])),\not-follow(lit("//"))})],{}),
[appl(regular(\iter-star(sort("WhitespaceOrComment"))),[appl(prod(label("whitespace",sort("WhitespaceOrComment")),[lex("Whitespace")],{}),
[appl(prod(lex("Whitespace"),[\char-class([range(9,10),range(12,13),range(32,32)])],{}),[char(10)])[@loc=|project://MissGrant/input/
missgrant.ctl|(42,1,<3,18>,<4,0>)]])[@loc=|project://MissGrant/input/missgrant.ctl|
(42,1,<3,18>,<4,0>)],appl(prod(label("whitespace",sort("WhitespaceOrComment")),[lex("Whitespace")],{}),[appl(prod(lex("Whitespace"),[\char-
class([range(9,10),range(12,13),range(32,32)])],{}),[char(32)])[@loc=|project://MissGrant/input/missgrant.ctl|(43,1,<4,0>,<4,1>)]])[@loc=|
project://MissGrant/input/missgrant.ctl|(43,1,<4,0>,<4,1>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(42,2,<3,18>,<4,1>)]])[@loc=|
project://MissGrant/input/missgrant.ctl|(42,2,<3,18>,<4,1>)],appl(prod(label("event",sort("Event")),
[label("name",lex("Id")),layouts("Standard"),label("token",lex("Id"))],{}),[appl(prod(lex("Id"),[conditional(seq([\char-
class([range(65,90),range(97,122)]),conditional(\iter-star(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-
follow(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)]))})]),{delete(keywords("Reserved"))})],{}),
[appl(regular(seq([\char-class([range(65,90),range(97,122)]),conditional(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-follow(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))})])),[char(108),appl(regular(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))),[char(105),char(103),char(104),char(116),char(79),char(110)])[@loc=|
project://MissGrant/input/missgrant.ctl|(45,6,<4,2>,<4,8>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(44,7,<4,1>,<4,8>)]])[@loc=|
project://MissGrant/input/missgrant.ctl|(44,7,<4,1>,<4,8>)],appl(prod(layouts("Standard"),[conditional(\iter-
star(sort("WhitespaceOrComment")),{\not-follow(\char-class([range(9,10),range(12,13),range(32,32)])),\not-follow(lit("//"))})],{}),
[appl(regular(\iter-star(sort("WhitespaceOrComment"))),[appl(prod(label("whitespace",sort("WhitespaceOrComment")),[lex("Whitespace")],{}),
[appl(prod(lex("Whitespace"),[\char-class([range(9,10),range(12,13),range(32,32)])],{}),[char(32)])[@loc=|project://MissGrant/input/
missgrant.ctl|(51,1,<4,8>,<4,9>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(51,1,<4,8>,<4,9>)]])[@loc=|project://MissGrant/input/
missgrant.ctl|(51,1,<4,8>,<4,9>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(51,1,<4,8>,<4,9>)],appl(prod(lex("Id"),
[conditional(seq([\char-class([range(65,90),range(97,122)]),conditional(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-follow(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))})]),{delete(keywords("Reserved"))})],{}),[appl(regular(seq([\char-
class([range(65,90),range(97,122)]),conditional(\iter-star(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-
follow(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)]))})])),[char(76),appl(regular(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))),[char(49),char(79),char(78)])[@loc=|project://MissGrant/input/missgrant.ctl|
(53,3,<4,10>,<4,13>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(52,4,<4,9>,<4,13>)]])[@loc=|project://MissGrant/input/missgrant.ctl|
(52,4,<4,9>,<4,13>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(44,12,<4,1>,<4,13>)],appl(prod(layouts("Standard"),[conditional(\iter-
star(sort("WhitespaceOrComment")),{\not-follow(\char-class([range(9,10),range(12,13),range(32,32)])),\not-follow(lit("//"))})],{}),
[appl(regular(\iter-star(sort("WhitespaceOrComment"))),[appl(prod(label("whitespace",sort("WhitespaceOrComment")),[lex("Whitespace")],{}),
[appl(prod(lex("Whitespace"),[\char-class([range(9,10),range(12,13),range(32,32)])],{}),[char(10)])[@loc=|project://MissGrant/input/
missgrant.ctl|(56,1,<4,13>,<5,0>)]])[@loc=|project://MissGrant/input/missgrant.ctl|
(56,1,<4,13>,<5,0>)],appl(prod(label("whitespace",sort("WhitespaceOrComment")),[lex("Whitespace")],{}),[appl(prod(lex("Whitespace"),[\char-
class([range(9,10),range(12,13),range(32,32)])],{}),[char(32)])[@loc=|project://MissGrant/input/missgrant.ctl|(57,1,<5,0>,<5,1>)]])[@loc=|
project://MissGrant/input/missgrant.ctl|(57,1,<5,0>,<5,1>)]])[@loc=|project://MissGrant/input/missgrant.ctl|(56,2,<4,13>,<5,1>)]])[@loc=|
project://MissGrant/input/missgrant.ctl|(56,2,<4,13>,<5,1>)],appl(prod(label("event",sort("Event")),
[label("name",lex("Id")),layouts("Standard"),label("token",lex("Id"))],{}),[appl(prod(lex("Id"),[conditional(seq([\char-
class([range(65,90),range(97,122)]),conditional(\iter-star(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-
follow(\char-class([range(48,57),range(65,90),range(95,95),range(97,122)]))})]),{delete(keywords("Reserved"))})],{}),
[appl(regular(seq([\char-class([range(65,90),range(97,122)]),conditional(\iter-star(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)])),{\not-follow(\char-
class([range(48,57),range(65,90),range(95,95),range(97,122)]))})])),[char(100),appl(regular(\iter-star(\char-
```

Abstract Syntax

```
data Controller
  = controller(list[Event] events,
               list[str] resets,
               list[Command] commands,
               list[State] states);

data State
  = state(str name,
          list[str] actions,
          list[Transition] transitions);

data Command    = command(str name, str token);
data Event      = event(str name, str token);
data Transition = transition(str event, str state);
```

Implode

```
controller(
  [
    event("doorClosed","D1CL"),
    event("drawerOpened","D2OP"),
    event("lightOn","L1ON"),
    event("doorOpened","D1OP"),
    event("panelClosed","PNCL")
  ],
  ["doorOpened"],
  [
    command("unlockPanel","PNUL"),
    command("lockPanel","PNLK"),
    command("lockDoor","D1LK"),
    command("unlockDoor","D1UL")
  ],
  [
    state(
      "idle",
      ["unlockDoor","lockPanel"],
      [transition("doorClosed","active")]),
    state(
      "active",
      [],
      [
        transition("drawerOpened","waitingForLight"),
        transition("lightOn","waitingForDrawer")
      ]),
    state(
      "waitingForLight",
      [],
      [transition("lightOn","unlockedPanel")]),
    state(
      "waitingForDrawer",
      [],
      [transition("drawerOpened","unlockedPanel")]),
    state(
      "unlockedPanel",
      ["unlockPanel","lockDoor"],
      [transition("panelClosed","idle")])
  ])
```

Pattern matching

```
int x := 3;

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

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

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

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

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern matching

type-based matching

```
int x := 3;

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

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

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

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

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern matching

type-based matching

structural matching

```
int x := 3;

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

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

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

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

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern matching

type-based matching

structural matching

anti-matching

```
int x := 3;

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

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

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

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

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern matching

type-based matching

structural matching

anti-matching

list matching

```
int x := 3;

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

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

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

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

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern 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};

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

3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern 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;
```

```
3 <- {1,2,3}
int x <- {1,2,3}
```

Pattern 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}
```

Backtracking contexts

```
rascal>for ([*x, *y] := [1,1,1,1,1,1]) println("<x> <y>");
[] [1,1,1,1,1,1]
[1] [1,1,1,1,1]
[1,1] [1,1,1,1]
[1,1,1] [1,1,1]
[1,1,1,1] [1,1]
[1,1,1,1,1] [1]
[1,1,1,1,1,1] []
```

Backtracking contexts

```
rascal>for ([*x, *y] := [1,1,1,1,1,1]) println("<x> <y>");
[] [1,1,1,1,1,1]
[1] [1,1,1,1,1]
[1,1] [1,1,1,1]
[1,1,1] [1,1,1]
[1,1,1,1] [1,1]
[1,1,1,1,1] [1]
[1,1,1,1,1,1] []
```

```
rascal>for ([*x, *y] := [1,1,1,1,1,1], x == y) println("<x> <y>");
[1,1,1] [1,1,1]
```

Set matching

```
rascal>for ({*x, *y} := {1,2,3,4}) println("<x> <y>");
{4,3,2,1} {}
{4,3,2} {1}
{4,3,1} {2}
{4,3} {2,1}
{4,2,1} {3}
{4,2} {3,1}
{4,1} {3,2}
{4} {3,2,1}
{3,2,1} {4}
{3,2} {4,1}
{3,1} {4,2}
{3} {4,2,1}
{2,1} {4,3}
{2} {4,3,1}
{1} {4,3,2}
{} {4,3,2,1}
```

Comprehensions

list

```
[ i | i <- [1..100], i % 2 == 0 ];
```

map

```
( i: i*i | i <- [1..10] );
```

set & relation

```
{ <i, i*i> | i <- [1..10] };
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

```
r<0>;

r<1,0>;

r["active"];

r+;

r*;

r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection

```
r<0>;

r<1,0>;

r["active"];

r+;

r*;

r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection

invert

```
r<0>;

r<1,0>;

r["active"];

r+;

r*;

r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection

```
r<0>;
```

invert

```
r<1,0>;
```

```
r["active"];
```

right image

```
r+;

r*;

r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection

```
r<0>;
```

invert

```
r<1,0>;
```

```
r["active"];
```

right image

transitive closure

```
r+;
r*;
r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection

```
r<0>;
```

invert

```
r<1,0>;
```

```
r["active"];
```

right image

transitive closure

```
r+;
```

transitive reflexive closure

```
r*;
```

```
r o r
```

Relational calculus

```
r = {
  <"active","waitingForDrawer">,
  <"idle","active">,
  <"unlockedPanel","idle">,
  <"waitingForLight","unlockedPanel">,
  <"active","waitingForLight">,
  <"waitingForDrawer","unlockedPanel">
  };
```

projection
```
r<0>;
```

invert
```
r<1,0>;
```

```
r["active"];
```
right image

transitive closure
```
r+;
```

transitive reflexive closure
```
r*;
```

```
r o r
```
relation composition

Transformation

Transformation in Rascal

- Functional programming :-)
- Type-preserving *visit*

Visit

- Similar to case-based match construct
- Visits all nodes of data structure
- Specify cases of interest only
    - “Structure shy”
- Bottom-up, top-down, innermost, outermost strategies

Print all state names

do something when visiting a state

```
visit (ast) {
  case state(str name, _, _): println(name);
}
```

Rewriting

Rewrite

```
Controller desugar(Controller ctl) {
  init = ctl.states[0].name;
  ctl = visit (ctl) {
    case state(n, as, ts) => state(n, as, ts + nts)
       when nts := [ transition(e, init) | e <- ctl.resets ]
  };
  ctl.resets = [];
  return ctl;
}
```

Side condition

Traversal strategies

default strategy

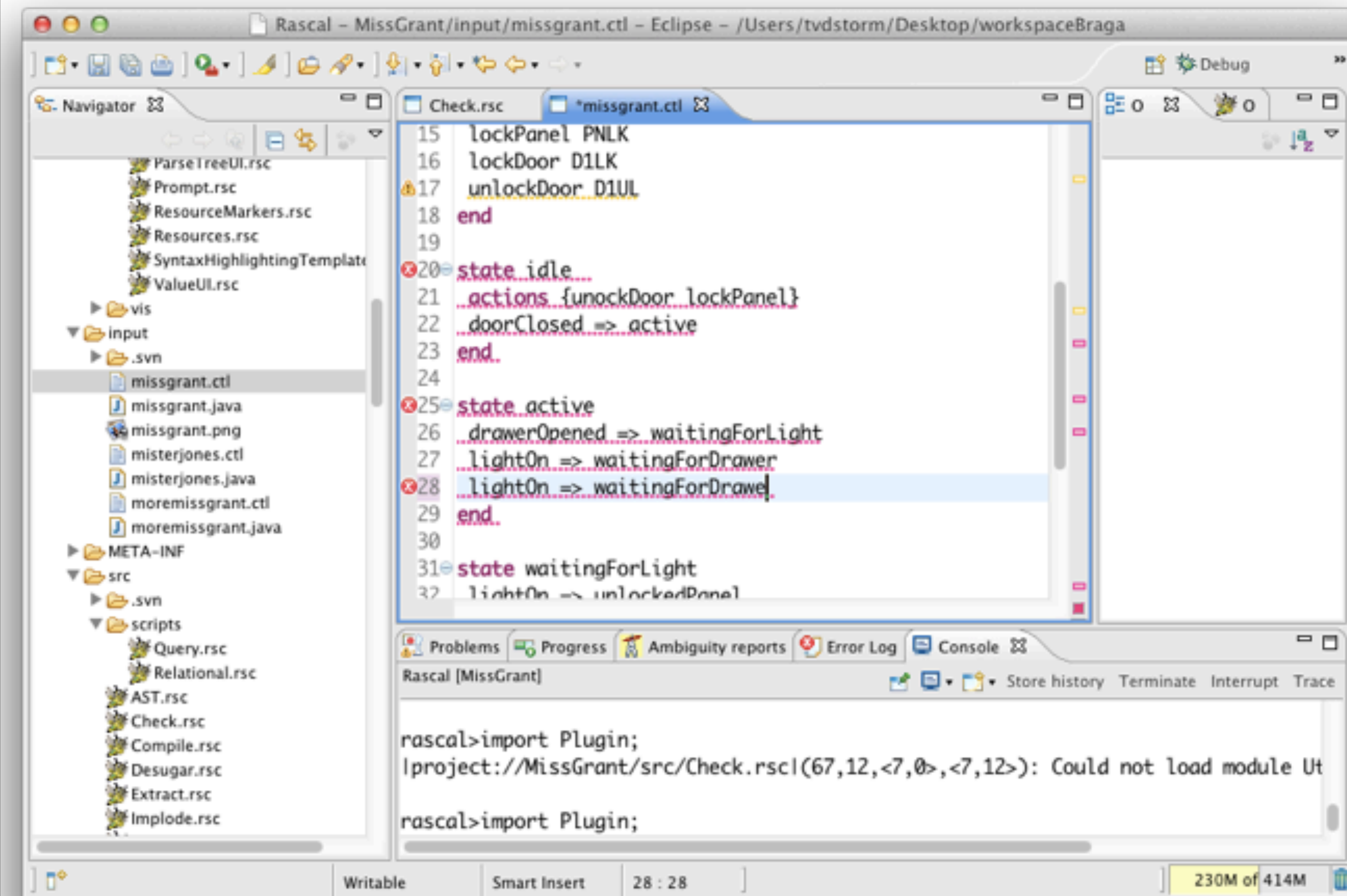
```
bottom-up visit (ast) {
  case state(str name, _, _): println(name);
}


top-down visit (ast) {
  case state(str name, _, _): println(name);
}
```

Analysis & transformation in DSL implementation

- Analysis: parsing, name resolution, type checking, model checking, etc.
- Transformation: desugaring, visualization, refactoring, optimization, compilation, etc.

524 SLOC



Concrete syntax

Abstract syntax

Unparse

Desugaring

Checking

Outline

Hyperlinking

Compilation

Visualization

Rename refactoring

Parallel merge

Concluding

- Functional programming for source code analysis and transformation
- Grammars, ADTs, pattern matching, comprehensions, relational calculus, ...
- Hooks into the Eclipse IDE
- Language workbench