A Snappy Introduction to Metaprogramming in Rascal

Red DevCon Winter: 26 January 2013

Vadim Zaytsev, SWAT, CWI

rascal

Joint work with (amongst others):

Bas Basten, Mark Hills, Anastasia Izmaylova, **Paul Klint**, Davy Landman, Arnold Lankamp, Bert Lisser, Atze van der Ploeg, Michael Steindorfer, **Tijs van der Storm**, **Jurgen Vinju**.

Software is complex

- The principles of software are easy
  - just a bunch of computer instructions
  - IO, arithmetic, control, done!
  - adv: patterns, concurrency, agile/formal — still easy
- The practice of software is incomprehensible
  - too much code
  - too much diversity
  - CPU is too fast
  - too much memory

Technical challenges

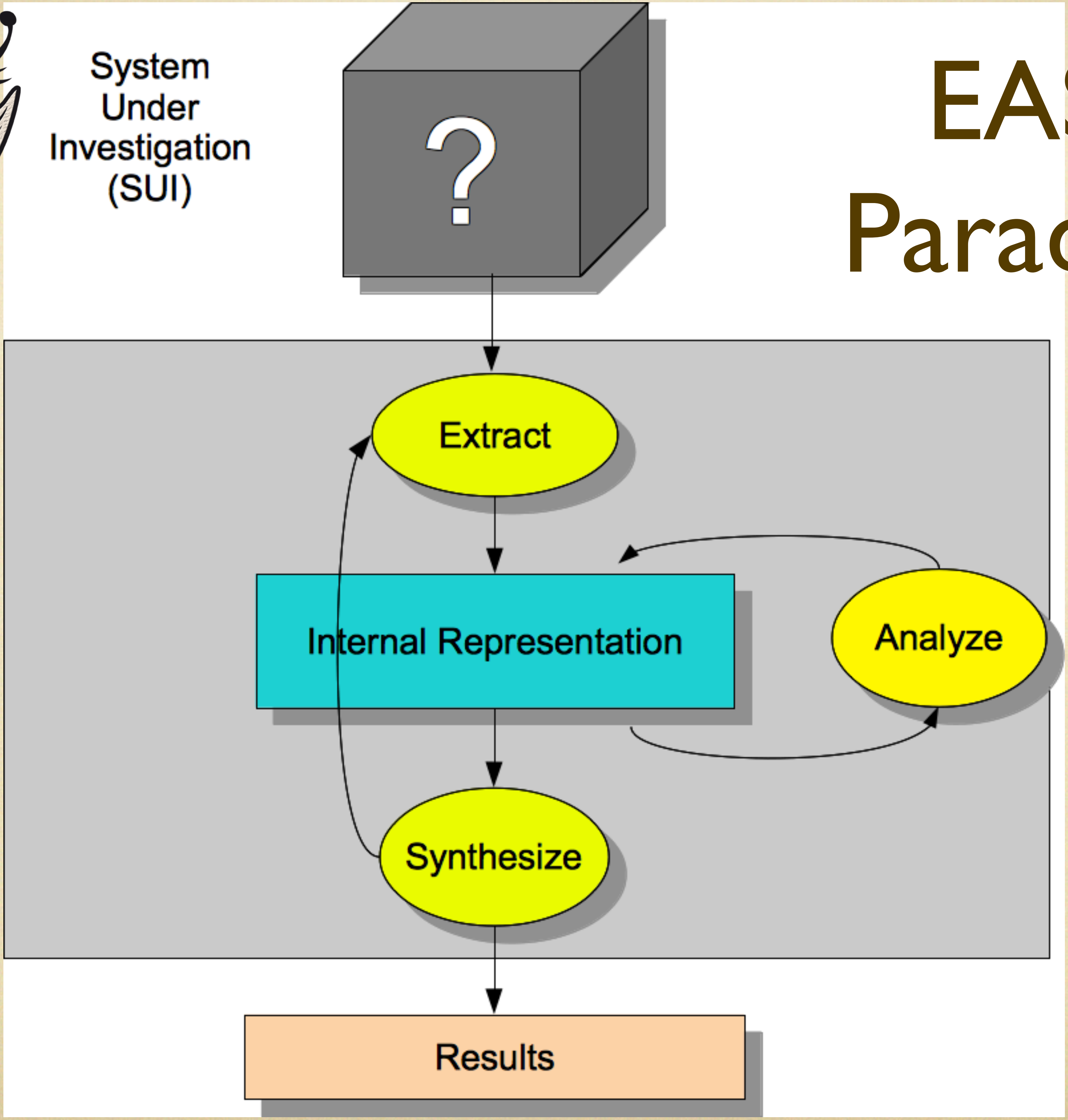
- How to parse source code/data files/models?
- How to extract facts from them?
- How to perform computations on these facts?
- How to generate new source code (transform, refactor, compile)?
- How to synthesize other information?

EASY: Extract-Analyze-SYnthesize Paradigm

Metaprogramming is EASY

- **Extract**
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- **Analyze**
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...
- **SYnthesize**
  - String templates
  - Concrete syntax
  - Interactive visualization generator

EASY Paradigm



Why a new language?

- No current technology spans the full range of **EASY** steps
- There are many fine technologies but they are
  - highly specialized with steep learning curves
  - hard to learn unintegrated technologies
  - not integrated with a standard IDE
  - hard to extend
- Goal: keep all benefits of **ASF+SDF** and **Rscript**
  - in a new, *unified*, *extensible*, *teachable* framework

Rascal keywords

- Complex built-in data types
- Immutable data
- Static safety
- Generic types
- Local type inference
- Pattern matching
- Syntax definitions & parsing
- Concrete syntax
- Visiting/traversal
- Comprehensions
- Higher-order
- Familiar syntax
- Java and Eclipse integration
- Read-Eval-Print (REPL)

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

Rascal design

- Java-like syntax

presumably familiar

- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

installs as a plugin

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

low barrier to entry,
learn features as you go

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

concrete syntax matching

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting

traversals, matching, …

- Relational calculus

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

relations for sharing/merging of facts for different languages

Rascal features

Rich (immutable) data

- Built-in sophisticated types:
  - lists
  - sets
  - maps
  - tuples
  - relations
- with comprehensions and many operators

```
rascal> [1..10]
list[int]: [1,2,3,4,5,6,7,8,9,10]
rascal> [x/2 | x <- [1..10]]
list[int]: [0,1,1,2,2,3,3,4,4,5]
rascal> {x/2 | x <- [1..10]} + {4,5,6}
set[int]: {6,5,4,3,2,1,0}
```

Syntax definitions

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/…
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

```
lexical Id = [A-Za-züäöß]+ !>> [A-Za-züäöß];
lexical Num = [0-9]+ !>> [0-9];
```

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/…
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

```
start syntax System = Line+;
syntax Line = Num ":" {Id ","}+ "." ;
```

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/…
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/…

layout WS = [\ \t\n\r]* !>> [\ \t\n\r];

- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Patterns

- Pattern matching
  - on concrete syntax
  - on lists
  - on sets
  - on trees
  - …
- Pattern-driven dispatch

```
rascal> {int x, str y} := {2}
bool: false
rascal> {int x, str y} := {2,"3"}
bool: true
rascal> {int x, *y, str z} := {2,2,2,"3",4,"2"}
bool: true
```

Other pattern kinds

- **Regular:** grep/Perl like regular expressions
  - /^<before:\W*><word:\w+><after:.*$>/
- **Abstract:** match data types
  - whileStat(Exp, Stats*)
- **Concrete:** match parse trees
  - ` while <Exp> do <Stats*> od `

Pattern-directed invocation

Prolog?

```
bool eqfp(fpnt(), fpnt()) = true;
bool eqfp(fpopt(), fpopt()) = true;
bool eqfp(fpplus(), fpplus()) = true;
bool eqfp(fpstar(), fpstar()) = true;
bool eqfp(fpempty(), fpempty()) = true;
bool eqfp(fpmany(L1), fpmany(L2)) = multiseteq(L1,L2);
default bool eqfp(Footprint pi, Footprint xi) = false;
```

Switch/case

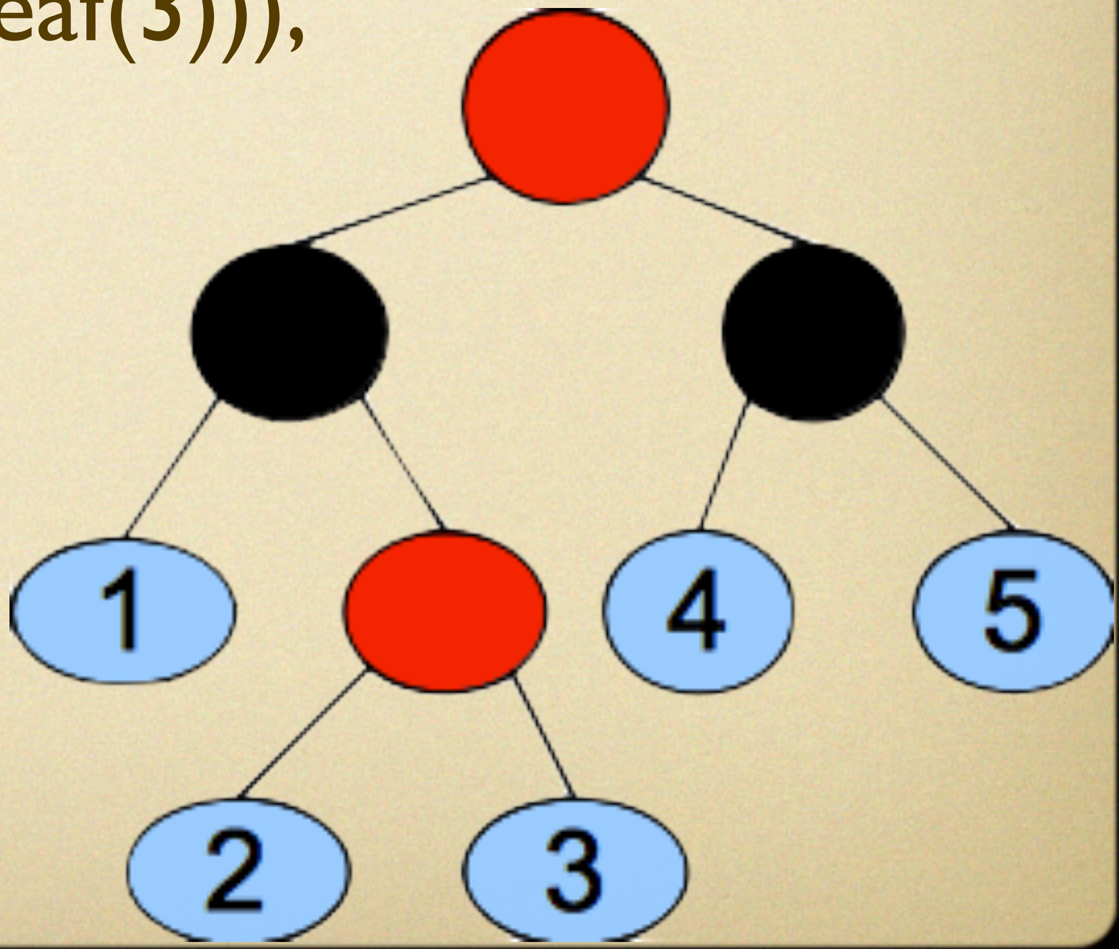
```
switch(p)
{
    case (DCGFun)`[]` => ["ε"];
    case (DCGFun)`<Word n>` =>
        ["<n>" | "<n>"==toLowerCase("<n>")];
    case (DCGFun)`(<{DCGFun ","}* args>)` =>
        [*getTags(a) | a <- args];
    case (DCGFun)`<Word f> (<{DCGFun ","}* as>)` =>
        ["<f>"] + [*getTags(a) | a <- as];
    default …
}
```

Visitor

```
@contributor{Bas Basten - Bas.Basten@cwi.nl (CWI)}
@contributor{Mark Hills - Mark.Hills@cwi.nl (CWI)}
module Operations

import AST;
import IO;

public Company cut(Company c) {
      return visit (c) {
            case employee(name, [*ep,ip:intProp("salary",salary),*ep2])
                => employee(name, [*ep,ip[intVal=salary/2],*ep2])
}}

public int total(Company c) {
return (0 | it+salary | /employee(name, [*ep,ip:intProp("salary",salary),*ep2]) <- c);
}
```

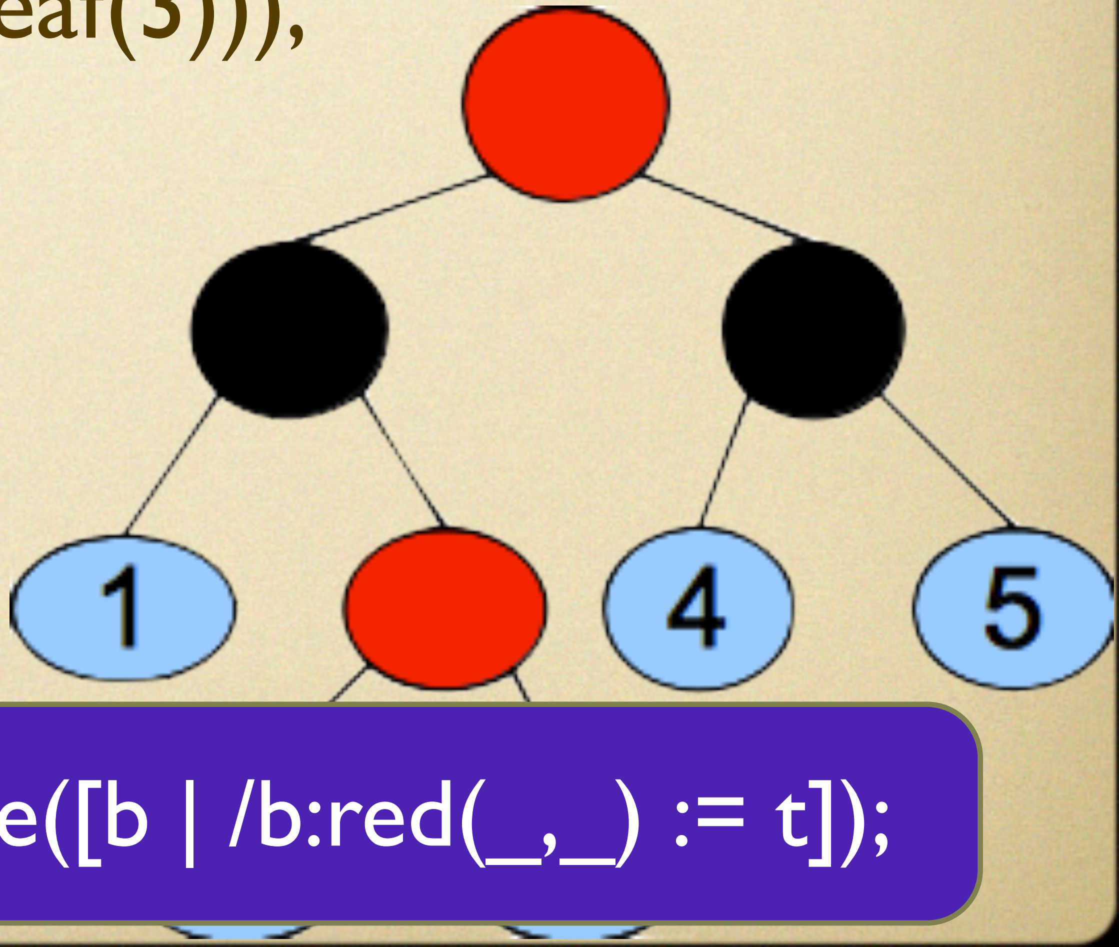
ADTs and visitors

```
data CTree =  leaf(int N)
           |  red(CTree left, CTree right)
           |  black(CTree left, CTree right) ;

rb = red(black(leaf(1), red(leaf(2), leaf(3))),
         black(leaf(4), leaf(5)));

public int cntRed(CTree t) {
  int c = 0;
  visit(t){case red(_,_): c += 1;};
  return c;
}
```

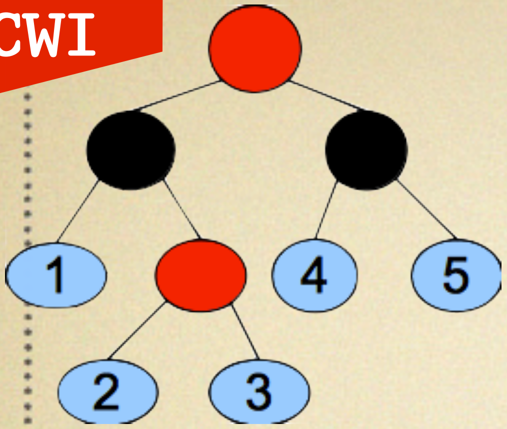


ADTs and visitors

```
data CTree =  leaf(int N)
            | red(CTree left, CTree right)
            | black(CTree left, CTree right) ;

rb = red(black(leaf(1), red(leaf(2), leaf(3))),
         black(leaf(4), leaf(5)));

public int cntRed(CTree t) {
  int c = 0;
  visit(t){case red(_,_): c += 1;};
  return c;
}
```

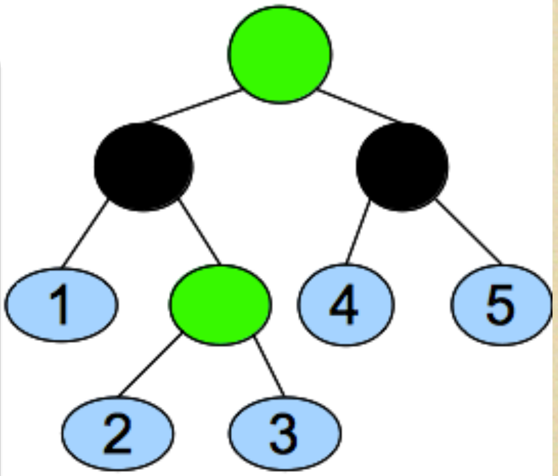


```
public int cnt2(CTree t) = size([b | /b:red(_,_) := t]);
```

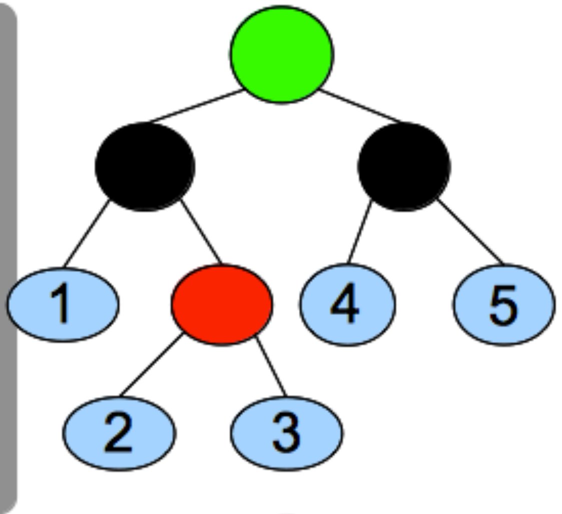


Full/shallow/deep

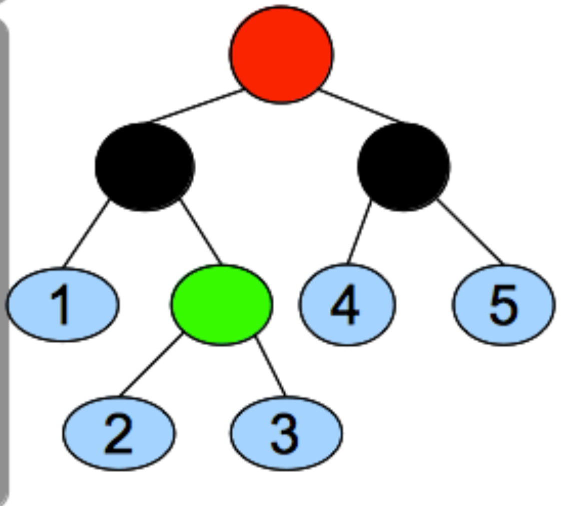
```
public CTree frepl(CTree T) {
   return visit (T) {
      case red(CTree T1, Ctree T2) =>  green(T1, T2)
   };
}
```



```
public Ctree srepl(CTree T) {
   return top-down-break visit (T) {
      case red(Ctree T1, CTree T2) =>  green(T1, T2)
   };
}
```

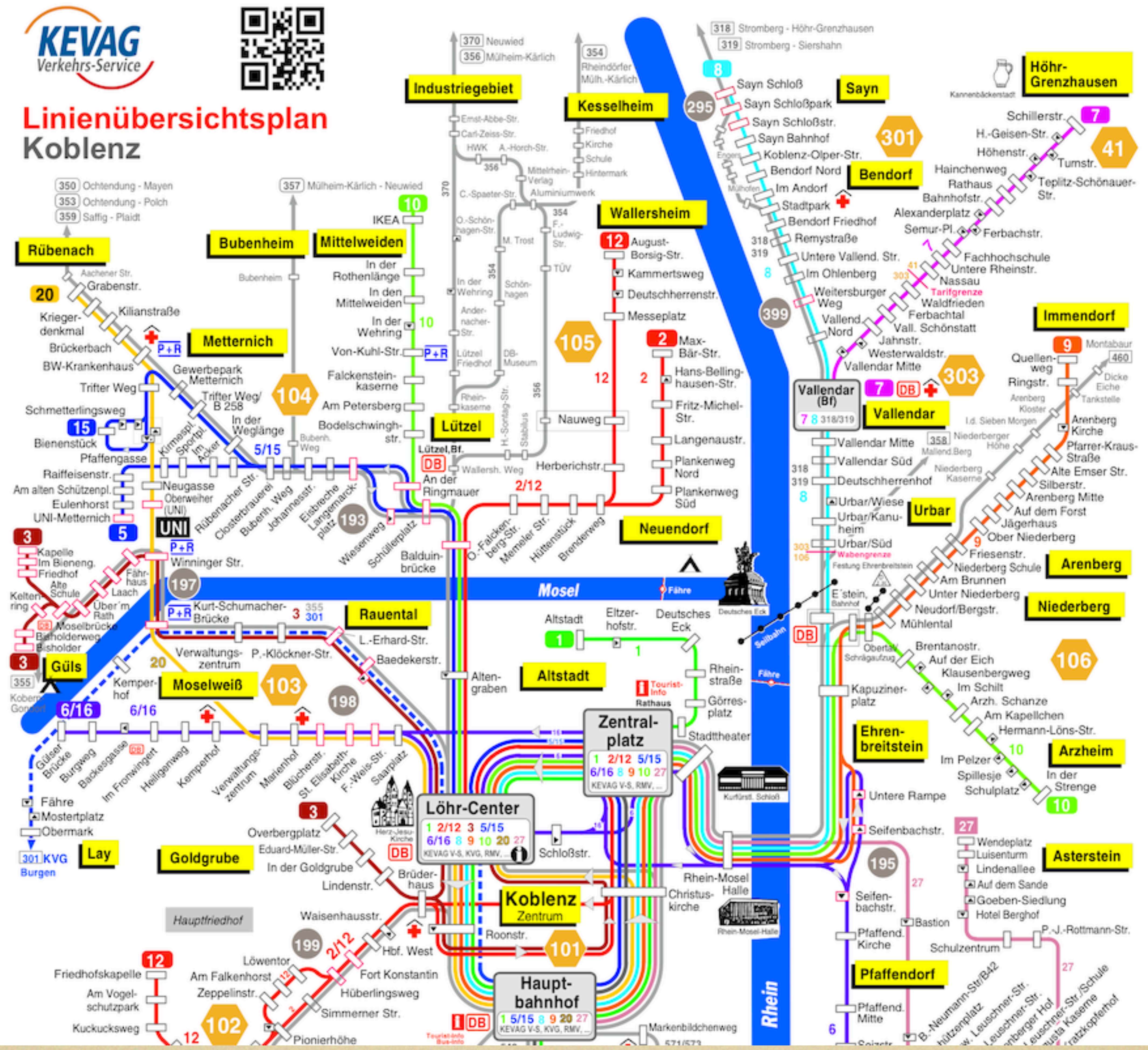


```
public Ctree drepl(Ctree T) {
   return bottom-up-break visit (T) {
      case red(CTree T1, CTree T2) =>  green(T1, T2)
   };
}
```



One-screen DSL

© koblenz.de; fair use claimed



http://www.vrminfo.de/linien-netze/liniennetzplan/koblenz

CWI

Bus lines in Koblenz

3: Overbergplatz, Eduard-Müller-Straße, In der Goldgrube, Lindenstraße, Brüderhaus, Roonstraße, Christuskirche, Löhr-Center, Ludwig-Erhard-Straße, Peter-Klöckner-Straße, Verwaltungszentrum, Kurt-Schumacher-Brücke, Winninger Straße.

5: Hauptbahnhof, Christuskirche, Zentralplatz, Löhr-Center, Balduinbrücke, Schüllerplatz, An der Ringmauer, Langemarckplatz, Johannesstraße, Bubenheimer Weg, Closterbrauerei, Rübenacher Straße, Im Acker, Sportplatz, Kirmesplatz, Raiffaisenstraße, Am Alten Schütenplatz, Eulenhorst, Oberweiher, Uni.

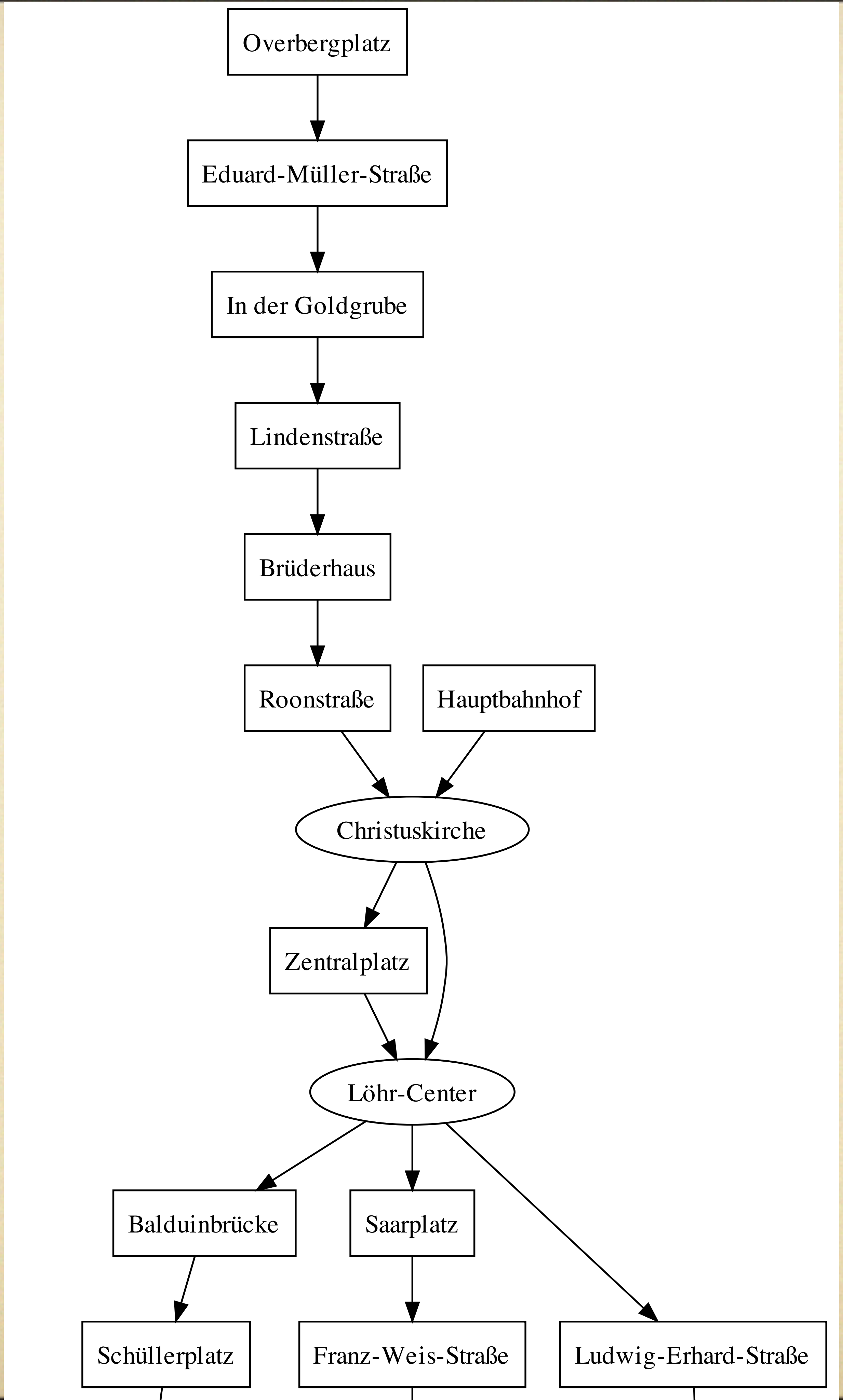
15: Hauptbahnhof, Christuskirche, Zentralplatz, Löhr-Center, Balduinbrücke, Schüllerplatz, An der Ringmauer, Langemarckplatz, Johannesstraße, Bubenheimer Weg, Closterbrauerei, In der Weglänge, Trifter Weg, Pfaffengasse, Bienenstück.

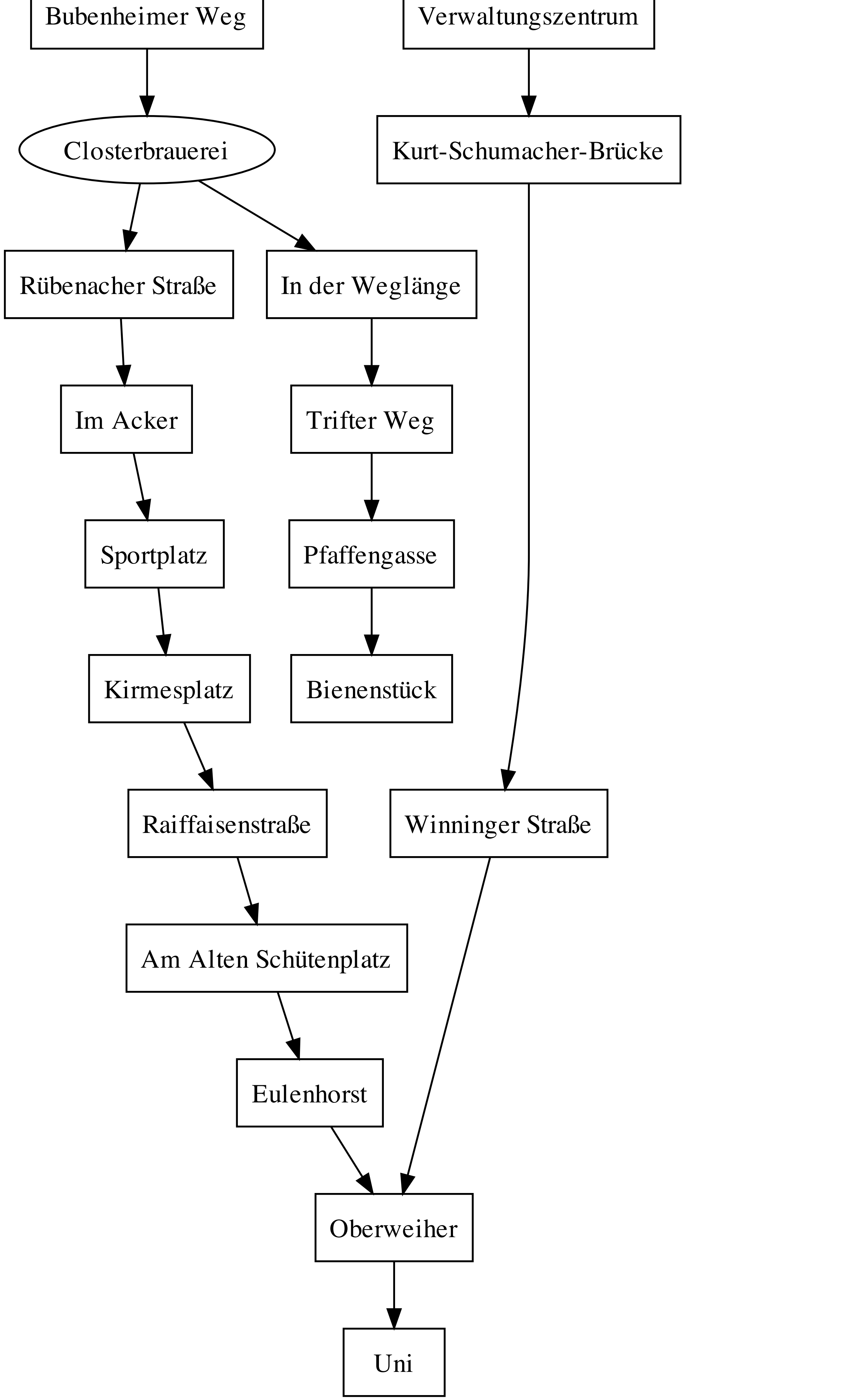
20: Hauptbahnhof, Christuskirche, Löhr-Center, Saarplatz, Franz-Weis-Straße, St-Elisabeth-Kirche, Blücherstraße, Marienhof, Verwaltungszentrum, Kurt-Schumacher-Brücke, Winninger Straße, Oberweiher.

SLPS/topics/implementation/busses/Koblenz.bus

```
start syntax System = Line+;
syntax Line = Num ":" {Id ","}+ "." ;
layout WS = [\ \t\n\r]* !>> [\ \t\n\r];
lexical Id = [A-Za-z][A-Za-züäöß\-\ ]+[A-Za-z] !>> [A-Za-z];
lexical Num = [0-9]+ !>> [0-9];

rel[Id,Id] extractGraph(loc source) = {<from,to> |
/Line b := parse(#start[System],source),
(Line)`<Num _>: <{Id ","}* _>, <Id from>, <Id to>, <{Id ","}* _>.` := b};

bool umsteigen(rel[Id,Id] sys, Id hs) = size(sys[hs]) > 1;

void synthesizeDotGraph(loc source, loc target)
{rel[Id from,Id to] conn = extractGraph(source);
 writeFile(target,
       "digraph Metro { node [shape=box]
       '<for (<from, to> <- conn) {>
       ' \"<from>\" -\> \"<to>\"<}>
       '<for (st <- conn<from>, umsteigen(conn, st)){>
       ' \"<st>\" [shape=ellipse]<}>
       '}");}
```

SLPS/topics/implementation/busses/src/Busses.rsc


Overbergplatz
Eduard-Müller-Straße
In der Goldgrube
Lindenstraße
Brüderhaus
Roonstraße
Hauptbahnhof
Christuskirche
Zentralplatz
Löhr-Center
Balduinbrücke
Saarplatz
Schüllerplatz
Franz-Weis-Straße
Ludwig-Erhard-Straße



Bubenheimer Weg

Verwaltungszentrum

Closterbrauerei

Kurt-Schumacher-Brücke

Rübenacher Straße

In der Weglänge

Im Acker

Trifter Weg

Sportplatz

Pfaffengasse

Kirmesplatz

Bienenstück

Raiffaisenstraße

Winninger Straße

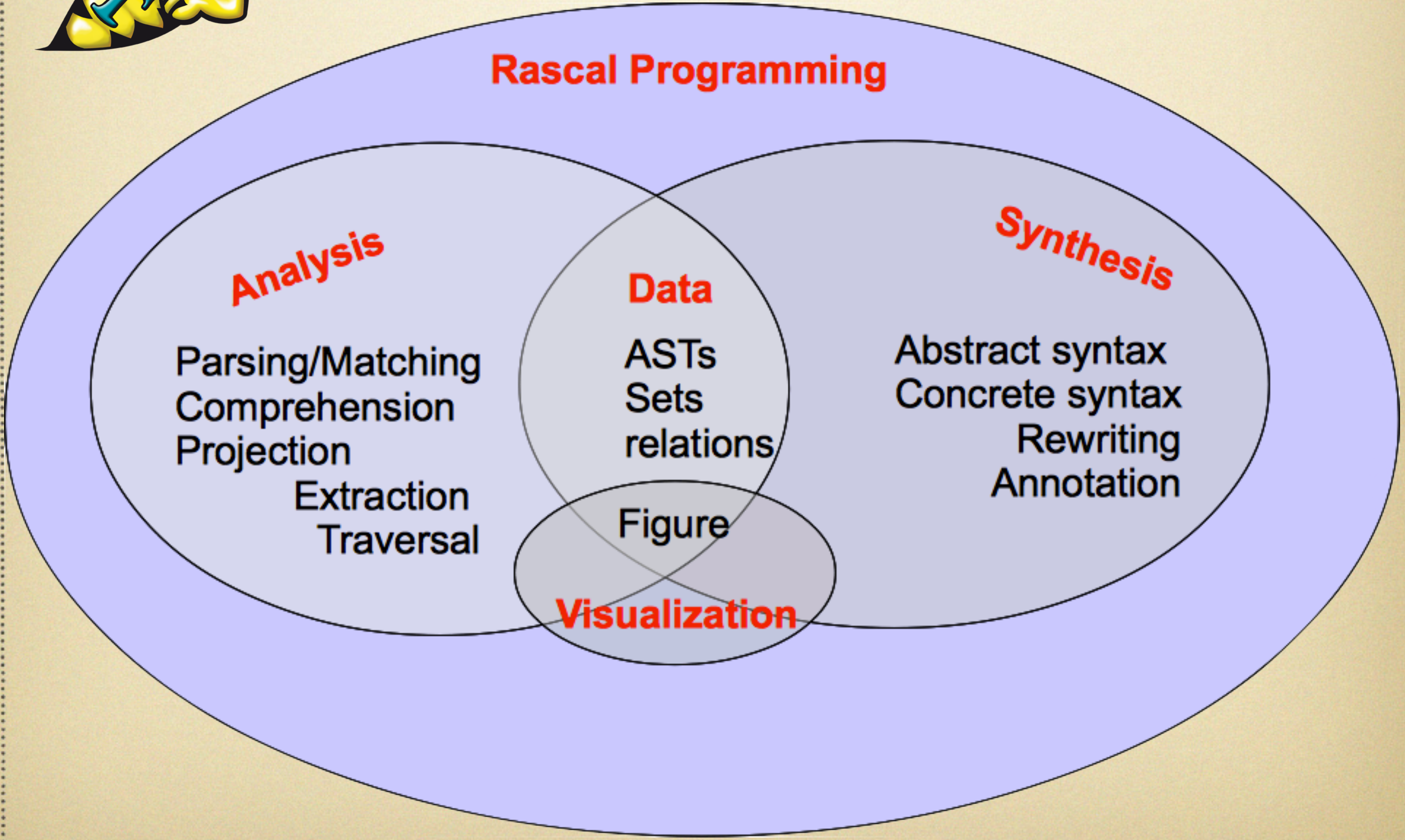
Am Alten Schütenplatz

Eulenhorst

Oberweiher

Uni

Bridging the gaps



Rascal Programming

Analysis

Parsing/Matching
Comprehension
Projection
Extraction
Traversal

Data

ASTs
Sets
relations

Synthesis

Abstract syntax
Concrete syntax
Rewriting
Annotation

Figure

Visualization

Benchmarking

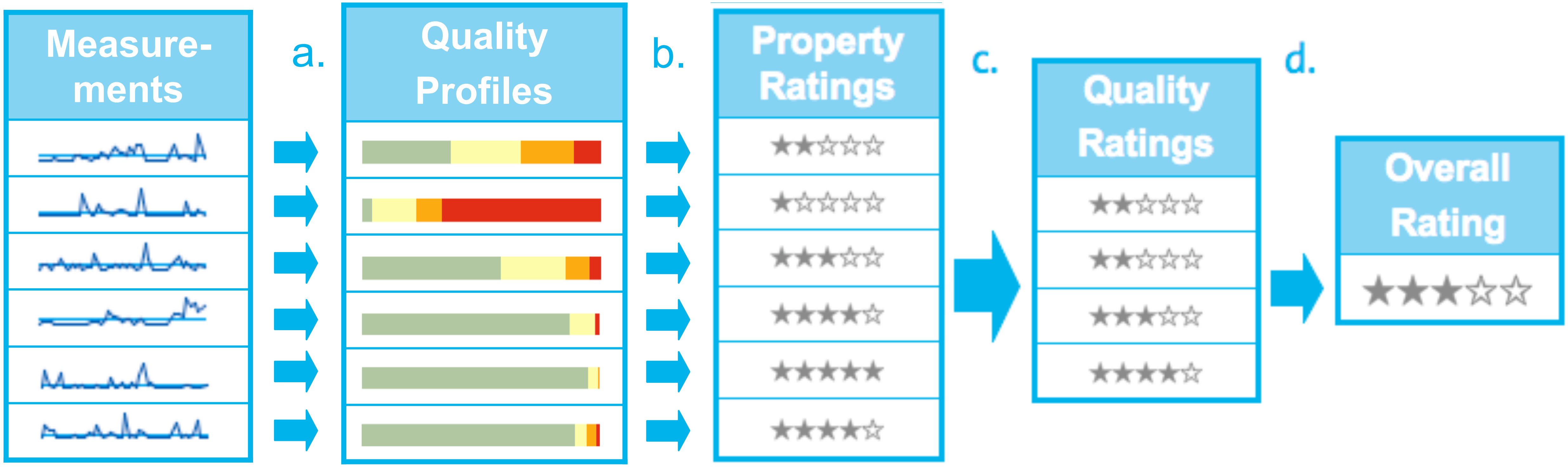
Software Improvement Group

http://sig.eu

A practical model for measuring maintainability

Heitlager, Kuipers, Visser in QUATIC 2007, IEEE Press

a. Aggregate measurements into “Quality Profiles”
b. Map measurements and quality profiles to ratings for system properties
c. Map ratings for system properties to ratings for ISO/IEC 9126 quality characteristics
d. Map to overall rating of technical quality



Measure-ments

a.

Quality Profiles

b.

Property Ratings

★★☆☆☆
★☆☆☆☆
★★★☆☆
★★★★☆
★★★★★
★★★★☆

c.

Quality Ratings

★★☆☆☆
★★☆☆☆
★★★☆☆
★★★★☆

d.

Overall Rating

★★★☆☆

vadim@grammarware.net

- http://rascal-mpl.org
- http://ask.rascal-mpl.org
- http://tutor.rascal-mpl.org

questions