

Software Language Engineering

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



Centrum Wiskunde & Informatica



university of
groningen

About me

- Senior researcher at CWI in the Software Analysis and Transformation team (SWAT)
- Professor software engineering at University of Groningen
- Expertise: language design, language engineering, language workbenches, ... META stuff ;)



Centrum Wiskunde & Informatica



**university of
groningen**



www.rascal-mpl.org

Outlook

- Lecture (Tuesdays, 13.00 ~ 15.00)
 - First part: Introduction to the course
 - Coffee break
 - Second part: Introduction to Rascal
- Lab/office hours (Thursdays, 11.00 ~ 13.00)
 - First hands-on with Rascal

Software Languages?

- Programming languages
- Domain-specific languages
- Data formats
- Specification languages
- Modeling languages

Software Languages?

- Programming languages
 - Java, C#, C, Ruby, Pascal, etc.
- Domain-specific languages
 - SQL, HTML, make, LaTeX
- Data formats
 - XML, CSV, JSON, YAML
- Specification languages
 - Event-B, Alloy, Promela
- Modeling languages
 - Modelica, UML, BPMN

Software Languages:
languages used for/in/during
software engineering

Software Language *Engineering*

- Engineering ~ maturity of a field of creating stuff
 - Repeatable, reliable, maintainable, performant language implementation
- Principled techniques
- Best practices
- Tools



SLE 2018

Software language engineering (SLE) is the discipline of engineering languages and their tools required for the creation of software. It abstracts from the differences between programming languages, modeling languages, and other software languages, and emphasizes the engineering facet of the creation of such languages, that is, the establishment of the scientific methods and practices that enable the best results.

Syntax

Semantics

Representation

Software
Language

Tooling

Analysis

Transformation

Topics of this course

- Concrete syntax: grammars, parsing
- Abstract syntax: data types, meta models
- Wellformedness: type and name checking
- Semantics: interpretation, simulation, evaluation
- Semantics: compilation, code generation
- Transformation: normalization and refactoring

Concrete syntax

```

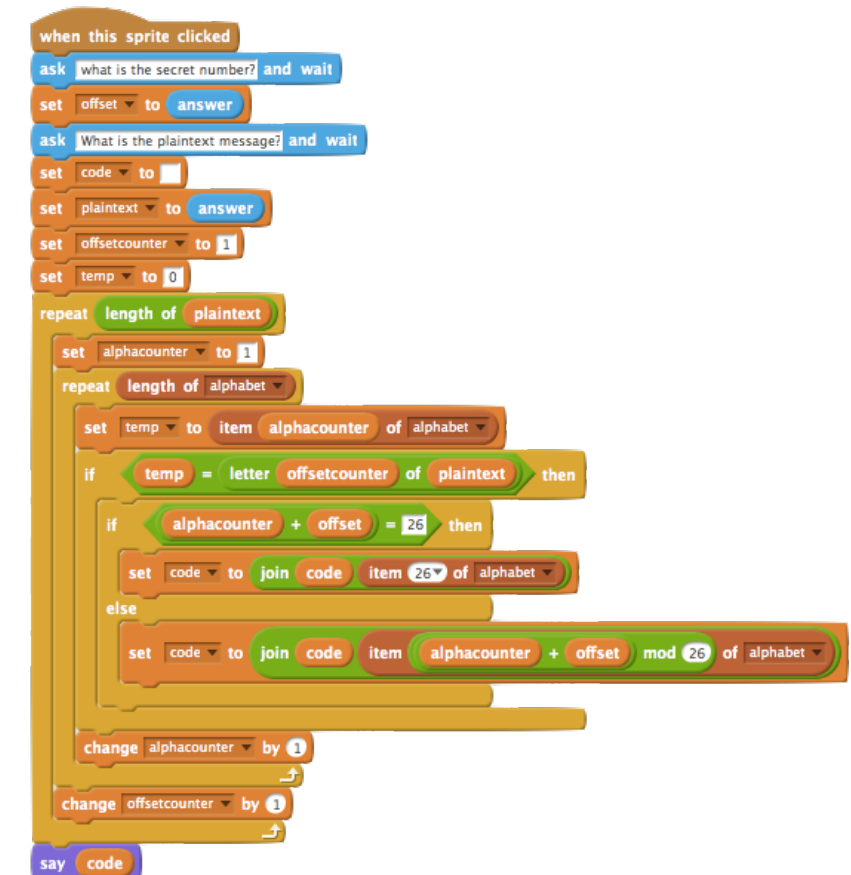
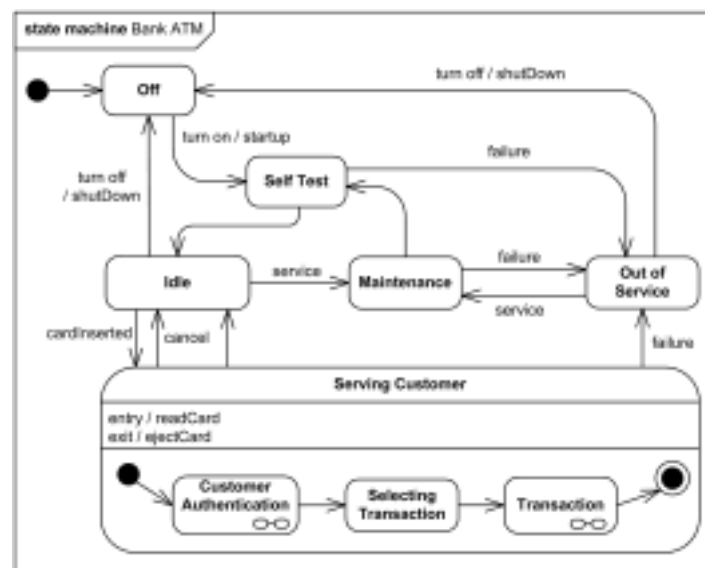
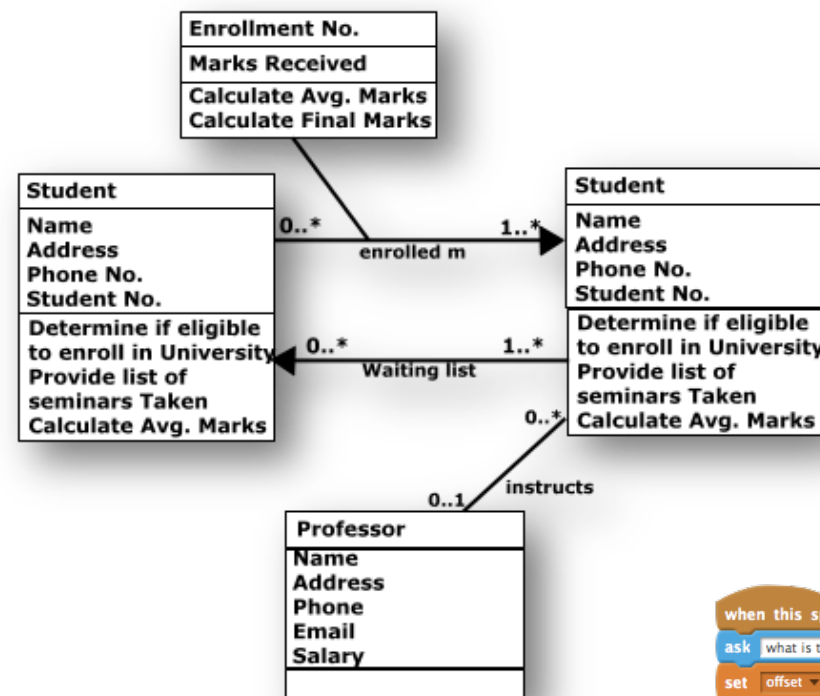
form taxOfficeExample {
  "Did you buy a house in 2010?"
  hasBoughtHouse: boolean

  "Did you enter a loan?"
  hasMaintLoan: boolean

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

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

```

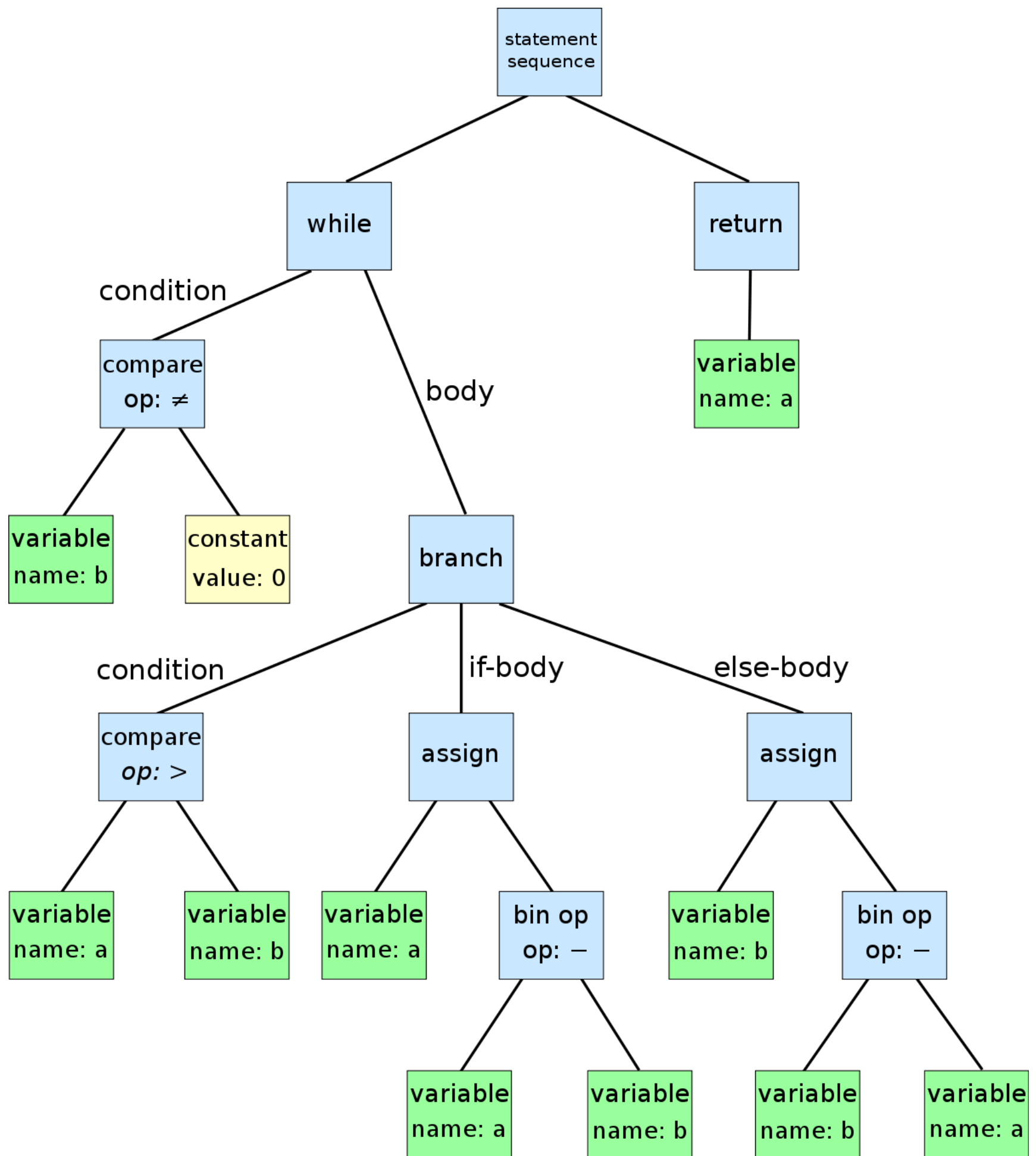


Defining Concrete Syntax

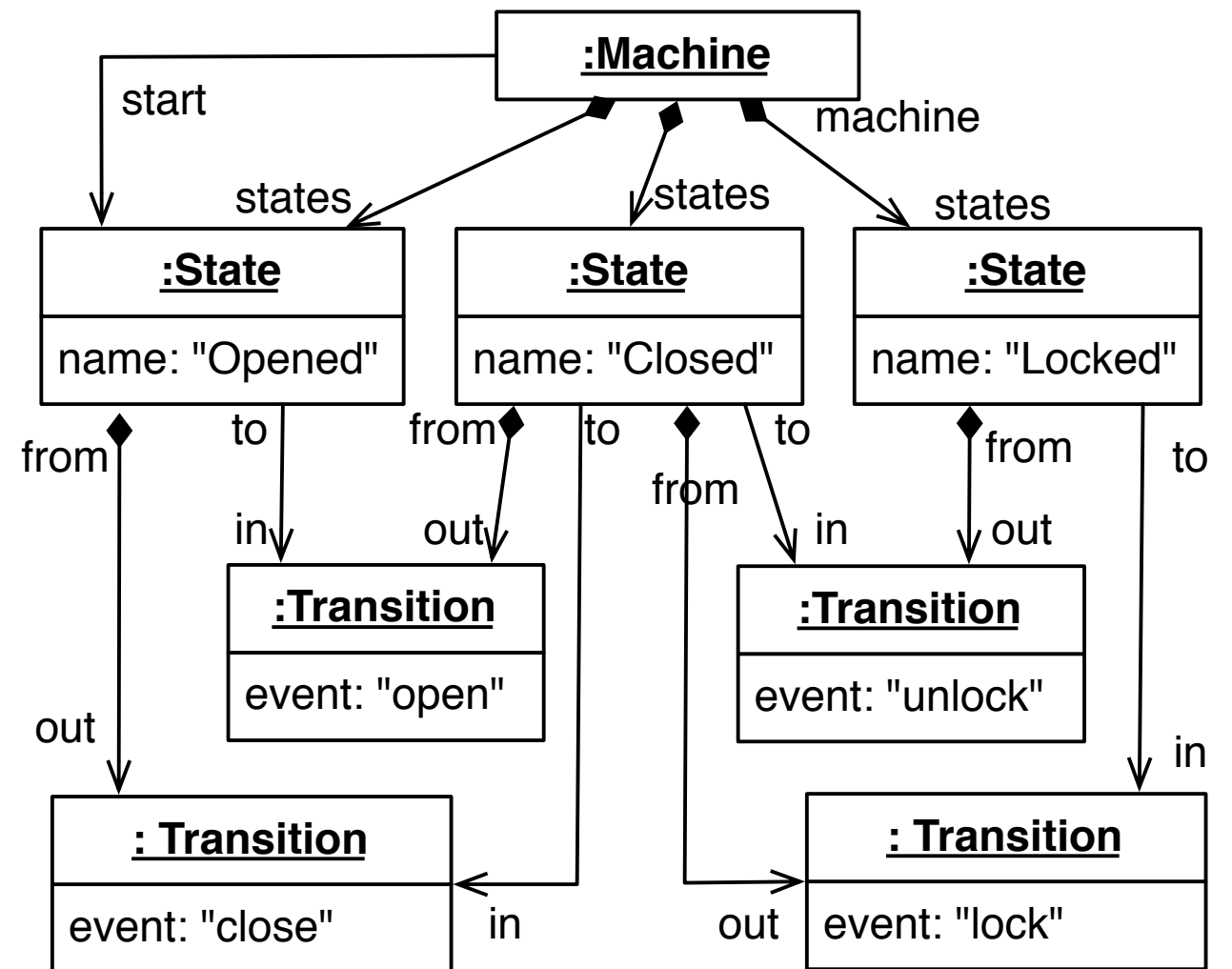
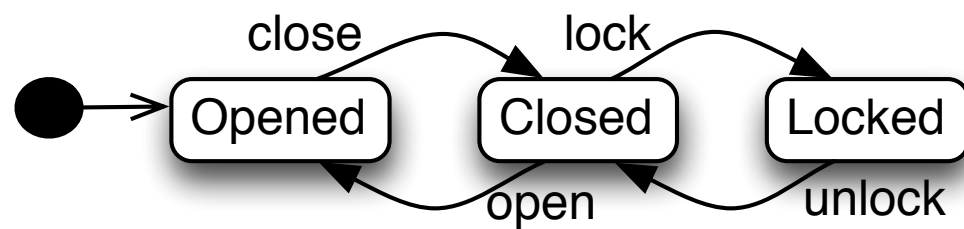
- Context-free grammars, parser generators, parser combinators, etc.
- Diagram editor frameworks (e.g., Eclipse Sirius)
- Structured editor frameworks (e.g., JetBrains MPS)

$$S \rightarrow AC|CB$$
$$C \rightarrow aCb|a|b$$
$$A \rightarrow aA|\epsilon$$
$$B \rightarrow Bb|\epsilon$$
$$S \rightarrow AC|CB$$
$$C \rightarrow aCb|\epsilon$$
$$A \rightarrow aA|\epsilon$$
$$B \rightarrow Bb|\epsilon$$

Abstract Syntax



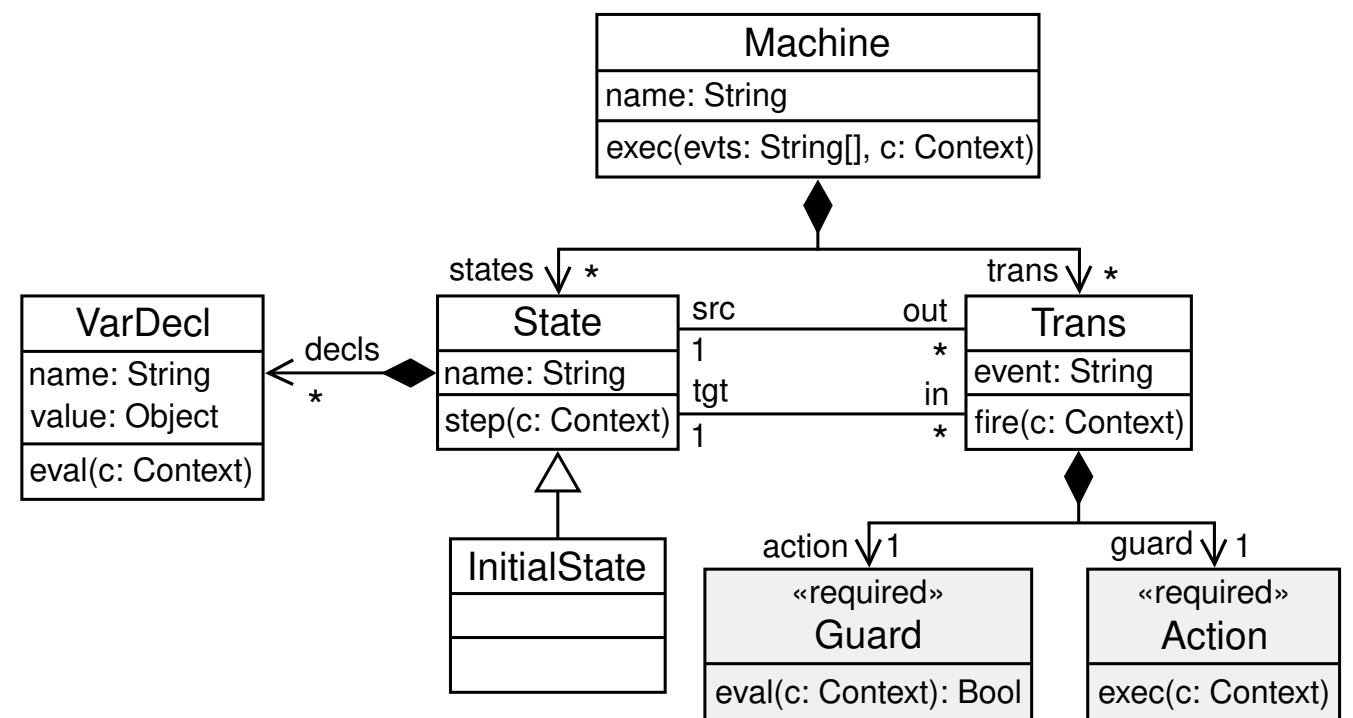
Abstract Syntax



Defining Abstract Syntax

- Algebraic Data Types
- Meta models (OO Class hierarchy)


```
data Expr
= var(Id id)
| integer(int intValue)
| string(str strValue)
| money(real realValue)
| \true()
| \false()
| not(Expr arg)
| mul(Expr lhs, Expr rhs)
| div(Expr lhs, Expr rhs)
| add(Expr lhs, Expr rhs)
| sub(Expr lhs, Expr rhs)
```



Wellformedness

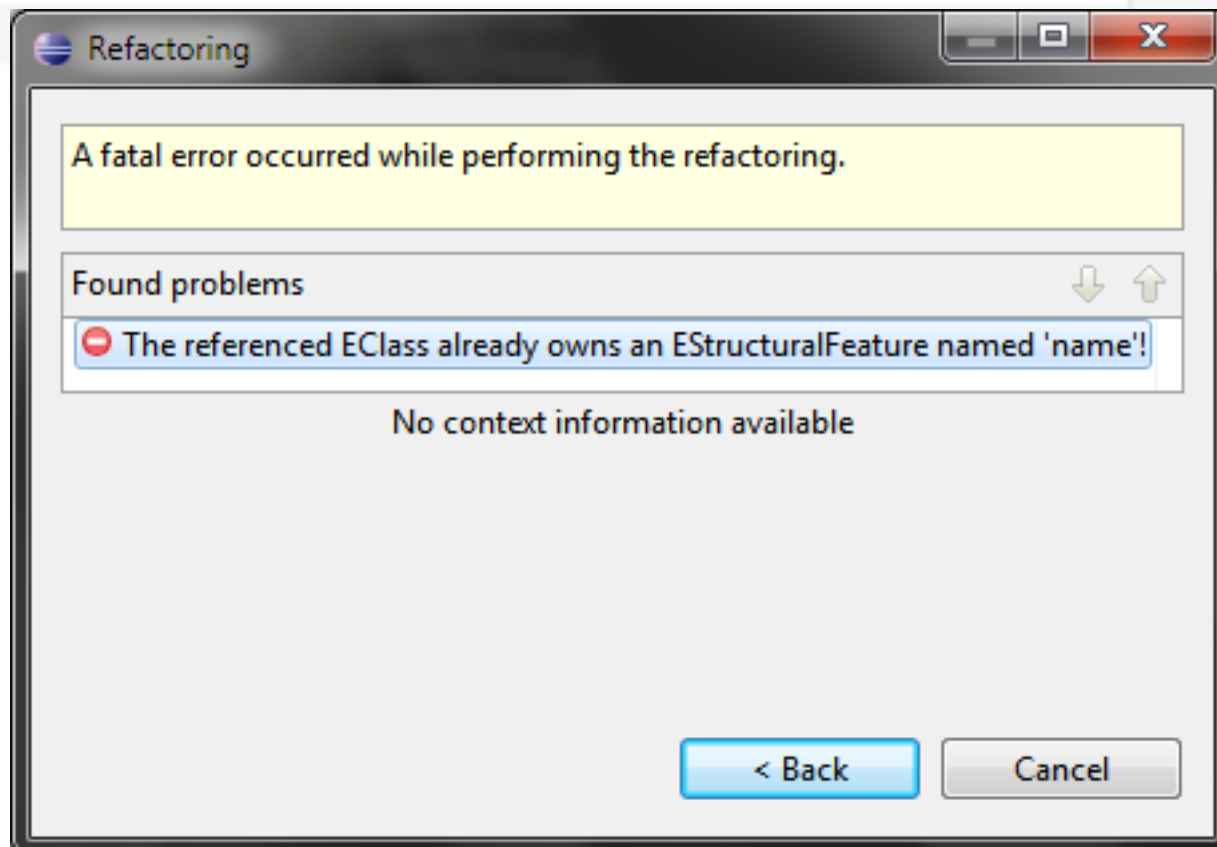
```
9 import UIKit
```

```
11 class ViewController: UIViewController {
```

```
13 override func viewDidLoad() {  Method does not override any method from its superclass  
14     super.viewDidLoad()  
15 }
```

```
17 override func didReceiveMemoryWarning() {  
18     super.didReceiveMemoryWarning()  
19 }
```

```
21 }
```



Wellformedness

- All checks outside realm of syntax.
- Reference checking: no dangling references, duplicate declarations etc.
- Type checking: are operations applied to operands of the right type?
- => static analysis, constraint checking, validation etc.

Semantics

$$\begin{array}{c}
 \frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)} \\
 \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1.field* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN-FIELD)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1[*]* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN-ARRAY)}
 \end{array}$$

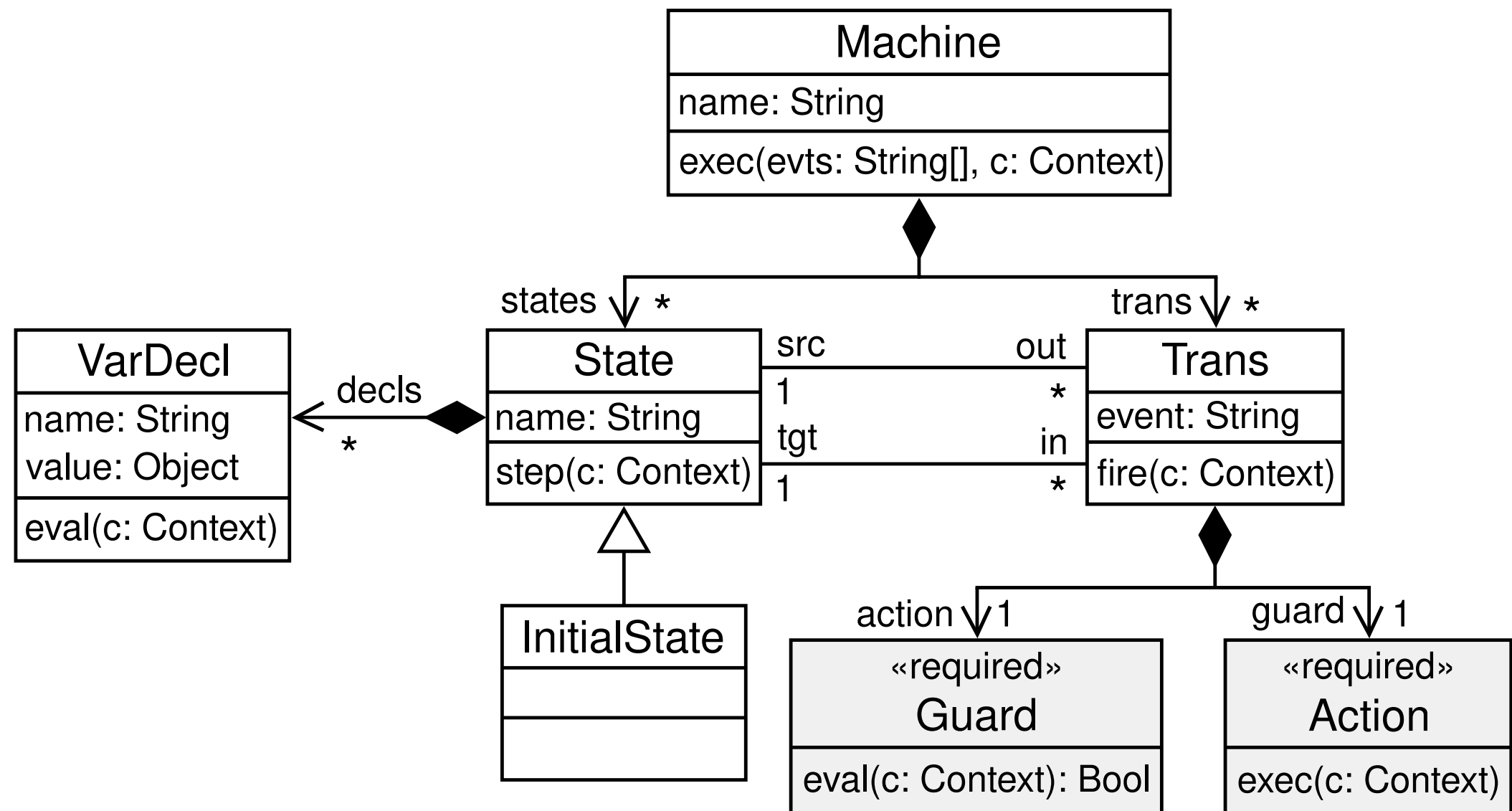
$$\frac{v \in \mathbf{R}; \text{pointer_type_p}(v); \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash v \oplus e \Rightarrow \mathbf{R}' \cup \{v \oplus e\}} \text{ (BINOP1)}$$

$$\frac{v \in \mathbf{R}'; \text{pointer_type_p}(v); \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash e \oplus v \Rightarrow \mathbf{R}' \cup \{e \oplus v\}} \text{ (BINOP2)} \quad \frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}'; \mathbf{R}' \vdash e_2 \Rightarrow \mathbf{R}''}{\mathbf{R} \vdash e_1; e_2 \Rightarrow \mathbf{R}''} \text{ (SEQ)}$$

$$\frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}_1; \mathbf{R}_1 \vdash e_2 \Rightarrow \mathbf{R}_2; \mathbf{R}_1 \vdash e_3 \Rightarrow \mathbf{R}_3; \mathbf{R}_2 \cup \mathbf{R}_3 \vdash e_4 \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash \text{if } (e_1) \text{ then } \{e_2\} \text{ else } \{e_3\} e_4 \Rightarrow \mathbf{R}'} \text{ (IF)}$$

$$\frac{\begin{array}{c} \text{function_type_p}(e) \\ \{e.args[i] \mid 0 < i < e.numargs \wedge \text{pointer_type_p}(e.args[i])\} \\ \cup \{v \mid \text{global_p}(v)\} \vdash e.body \Rightarrow \mathbf{R} \end{array}}{\{\} \vdash e \Rightarrow \mathbf{R}} \text{ (FUNCTION)}$$

Interpretation



Interpretation

```
int eval0(nat(int nat), PEnv penv) = nat;
```

```
int eval0(mul(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) * eval0(rhs, penv);
```

```
int eval0(div(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) / eval0(rhs, penv);
```

```
int eval0(add(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) + eval0(rhs, penv);
```

```
int eval0(sub(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) - eval0(rhs, penv);
```

```
int eval0(gt(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) > eval0(rhs, penv) ? 1 : 0;
```

```
int eval0(lt(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) < eval0(rhs, penv) ? 1 : 0;
```

```
int eval0(geq(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) ≥ eval0(rhs, penv) ? 1 : 0;
```

```
int eval0(leq(Exp lhs, Exp rhs), PEnv penv) = eval0(lhs, penv) ≤ eval0(rhs, penv) ? 1 : 0;
```

Compilation

```
Instrs compileStat(whileStat(EXP exp, list[STM] stats1)) {  
    entryLab = nextLabel();  
    endLab = nextLabel();  
    return [label(entryLab),  
            *compileExp(exp),  
            gofalse(endLab),  
            *compileStats(stats1),  
            go(entryLab),  
            label(endLab)];  
}
```

Code generation

```
str question2widget(Label l, Id v, QType t, str parent, str e)
= "var <v.name> = new QLrt.SimpleFormElementWidget({
  ' name: \"<v.name>\",
  ' label: <l.label>,
  ' valueWidget: new QLrt.<type2widget(t)>(<e>)
  '}).appendTo(<parent>);"

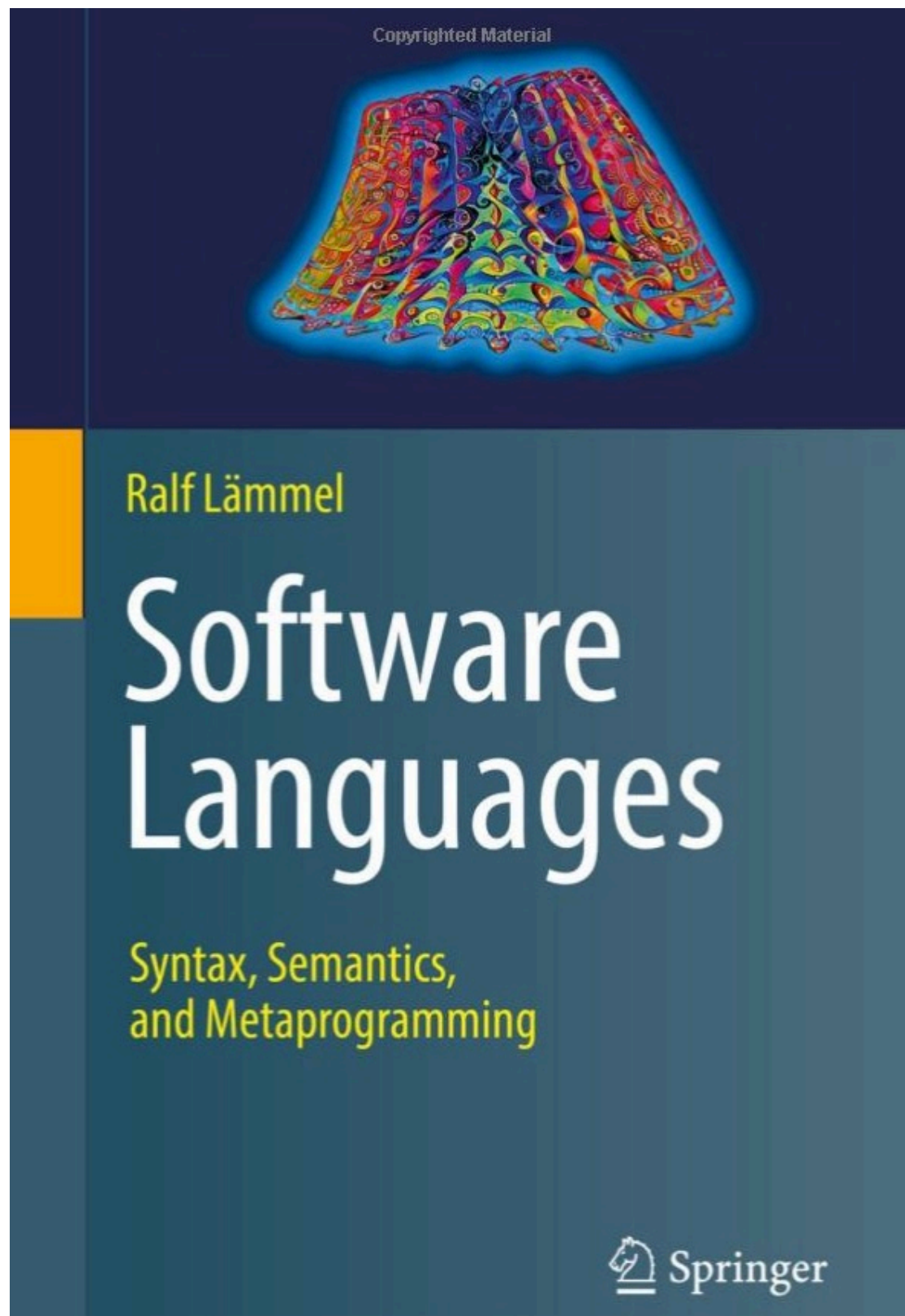
str exp2lazyValue(Expr e)
= "new QLrt.LazyValue(
  ' function () { return [<ps>]; },
  ' function (<ps>) { return <expr2js(e)>; }
  ')"

when str ps := expParams(e);
```

Transformations everywhere

- Parsing, deserialization/unparsing serialization
- Compilation, generation, translation
- Optimization, normalization, simplification
- Refactoring, reengineering, visualization

Course Organisation



Ralf Lämmel

FREE: <https://rug.on.worldcat.org/oclc/1036771828>

<https://cwi-swat.github.io/sle-rug/>

- Week 1: Introduction (Chapters 1 & 2)
- Week 2: Concrete syntax (Chapters 6 & 7)
- Week 3: Abstract syntax (Chapters 3 & 4)
- Week 4: Checking (Chapter 9)
- Week 5: Interpretation (Chapters 5 & 8)
- Week 6: Code generation (Chapter 5)
- Week 7: Transformation (Chapters 5 & 12)
- Week 8: Wrap up & grading of lab starts

Lab exercises: QL

```
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  
    }  
}
```

- ✓ 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

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

```

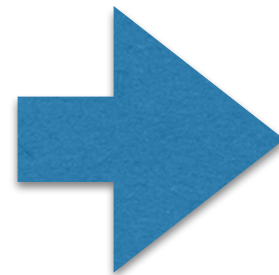
form taxOfficeExample {
  "Did you sell a house in 2010?"
  hasSoldHouse: boolean

  "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

```

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
  }
}

```

- Concrete syntax
- Abstract syntax
- Name resolution
- Type checking
- Interpretation
- Code generation
- Normalization
- Rename refactoring

- Week 2: Concrete syntax of QL using Rascal's grammar formalism (module `Syntax`)
- Week 3: Abstract syntax and name analysis of QL (modules `AST`, `CST2AST` and `Resolve`)
- Week 4: Type checker for QL (module `Check`)
- Week 5: Interpreter for QL (module `Eval`)
- Week 6: Code generator compiling QL to executable HTML and Javascript (module `Compile`)
- Week 7: Normalization of QL and rename refactoring (module `Transform`)
- Week 8: Grading of lab starts

Practicalities

- Work **individually**
- **Fork** the repo on the website (github!)
- You can help each other, but no copy-paste!!!
- No free-riding: the lab exercises are part of the material of the individual exam.
- Observe basic code quality guide lines (indentation, functional decomposition, modularity, ...)

How to pass the course

- Requirements
 - Complete the lab exercises (pass/fail)
 - Pass the exam
- Advice
 - Attend lectures
 - Use office hours (Thursday) for Q&A

Next ~45 minutes

- Introduction to Rascal
- Then start “Rascal Wax-on, Wax-off” tutorial