Context-oriented Language Engineering

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

(Context-oriented Language) Engineering

Context-oriented (Language Engineering)

Language Engineering

- Software Languages: specification, modeling, programming, DSLs, APIs, schemas, etc.

- “Engineering”: principled tools, formalisms, and techniques to implement such languages.

- My focus: Domain-specific Languages, aka “little languages”

Context-oriented Programming

COP treats context explicitly, and provides mechanisms to dynamically adapt behavior in reaction to changes in context, even after system deployment at runtime.

Context-oriented Programming is concerned with programming language constructs to represent and manipulate behavioral variations.

Hirschfeld, et al. *Context-Oriented Programming*, JOT, Vol 7, No 3, 2008.

Context-oriented Language Engineering (“COLE”)

COLE treats context explicitly, and provides mechanisms to dynamically adapt behavior in reaction to changes in context, even after **language** deployment at runtime.

COLE is concerned with **meta** programming language constructs to represent and manipulate **linguistic** variations.

Plan

- Broad view of language context: why context is important
- Zooming in on techniques for modular composition of interpreters with context propagation.

There's more to languages than syntax and semantics

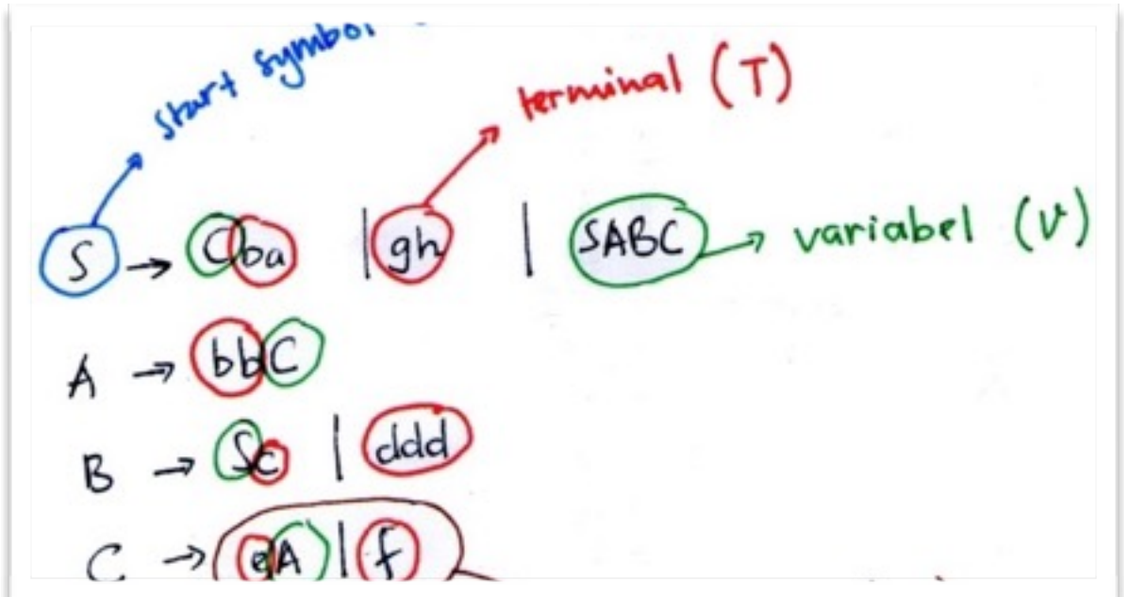


Table 11.1 (continued): *Semantic Functions*

$\mathcal{G}$: Cmd→U→G

$\mathcal{G}[\![\Phi]\!]\rho$ = a given γ associated with Φ (11.3)

$\mathcal{G}[\![\text{dummy}]\!]\rho = I$ (11.4)

$\mathcal{G}[\![\text{if E then } \Gamma_1 \text{ else } \Gamma_2]\!]\rho\theta =$
$\quad \mathcal{E}[\![E]\!]\rho(Cond\ \mathcal{G}[\![\Gamma_1]\!]\rho\theta, \mathcal{G}[\![\Gamma_2]\!]\rho\theta)$ (11.5)

$\mathcal{G}[\![\Gamma_1;\Gamma_2]\!]\rho = \mathcal{G}[\![\Gamma_1]\!]\rho \circ \mathcal{G}[\![\Gamma_2]\!]\rho$ (11.6)
(So $\mathcal{G}[\![\Gamma_1;\Gamma_2]\!]\rho\theta = \mathcal{G}[\![\Gamma_1]\!]\rho(\mathcal{G}[\![\Gamma_2]\!]\rho\theta)$.)

$\mathcal{G}[\![\text{while E do } \Gamma]\!]\rho\theta =$
$\quad fix(\lambda\theta'.\ \mathcal{E}[\![E]\!]\rho(Cond(\mathcal{G}[\![\Gamma]\!]\rho\theta', \theta)))$ (11.7)

$\mathcal{G}[\![\text{let I=E in } \Gamma]\!]\rho\theta =$
$\quad \mathcal{E}[\![E]\!]\rho(\lambda\delta.\mathcal{G}[\![\Gamma]\!](\rho[\delta/I])\theta)$ (11.8)

$\mathcal{G}[\![\text{goto E}]\!]\rho\theta = \mathcal{E}[\![E]\!]\rho(Jump)$ (11.9)
where $Jump(\varepsilon) = \varepsilon \in C \to (\varepsilon|C), Wrong$

$\mathcal{G}[\![\text{begin } I_1:\Gamma_1;I_2:\Gamma_2;\ldots;I_{n-1}:\Gamma_{n-1};I_n:\Gamma_n \text{ end}]\!]\rho =$
$(fix(\lambda\langle\theta_1,\theta_2,\ldots,\theta_{n-1},\theta_n\rangle.$
$\langle \mathcal{G}[\![\Gamma_1]\!]\rho'\theta_2, \mathcal{G}[\![\Gamma_2]\!]\rho'\theta_3, \ldots, \mathcal{G}[\![\Gamma_{n-1}]\!]\rho'\theta_n, \mathcal{G}[\![\Gamma_n]\!]\rho'\theta\rangle$
where $\rho'\ \rho[\theta_1 \text{ in } D/I_1][\theta_2 \text{ in } D/I_2]\ldots[\theta_n \text{ in } D/I_n]$))↓1 (11.10)

$\mathcal{E}$: Exp→U→W

$\mathcal{E}[\![I]\!]\rho\kappa = (\rho[\![I]\!])=?\to Wrong, \kappa(\rho[\![I]\!])$ (11.11)

$\mathcal{E}[\![\Pi]\!]\rho$ = a given ω associated with Π (11.12)

$\mathcal{E}[\![\text{true}]\!]\rho\kappa = \kappa(true \text{ in } E)$ (11.13)

$\mathcal{E}[\![\text{false}]\!]\rho\kappa = \kappa(false \text{ in } E)$ (11.14)

$\mathcal{E}[\![\text{if } E_0 \text{ then } E_1 \text{ else } E_2]\!]\rho\kappa =$
$\quad \mathcal{E}[\![E_0]\!]\rho(Cond(\mathcal{E}[\![E_1]\!]\rho\kappa, \mathcal{E}[\![E_2]\!]\rho\kappa))$ (11.15)

$\mathcal{E}[\![\text{let I=E}_0 \text{ in } E_1]\!]\rho\kappa =$
$\quad \mathcal{E}[\![E_0]\!]\rho(\lambda\delta.\ \mathcal{E}[\![E_1]\!](\rho[\delta/I])\kappa)$ (11.16)

What is language context?

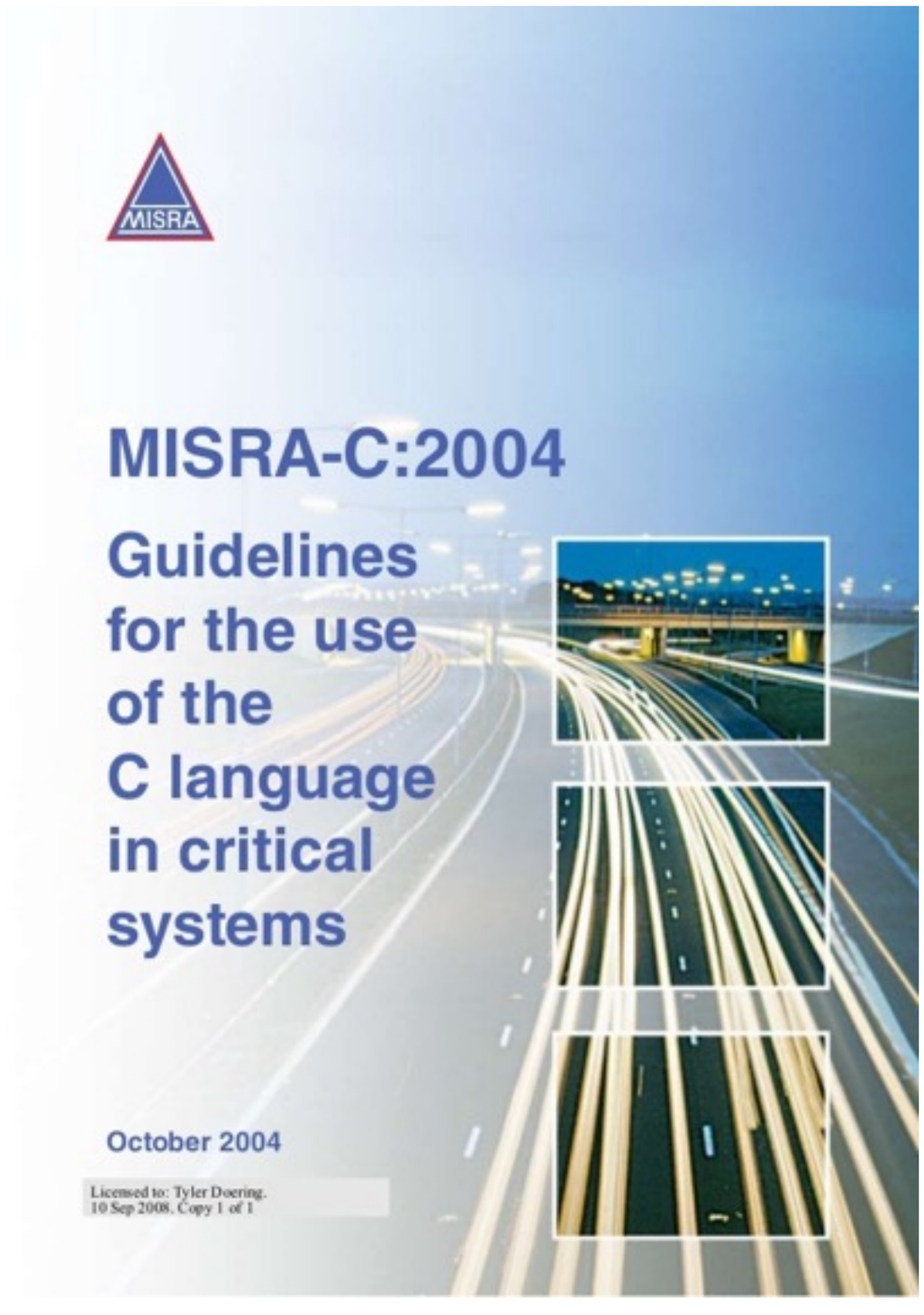
- Problem domain (What)
- Programmer: personalization, audience (Who)
- Programming activity: team, versioning, testing, debugging, specification, documenting, etc. (When)
- Device, server/client, battery/power, etc. (Where)

Linguistic variations

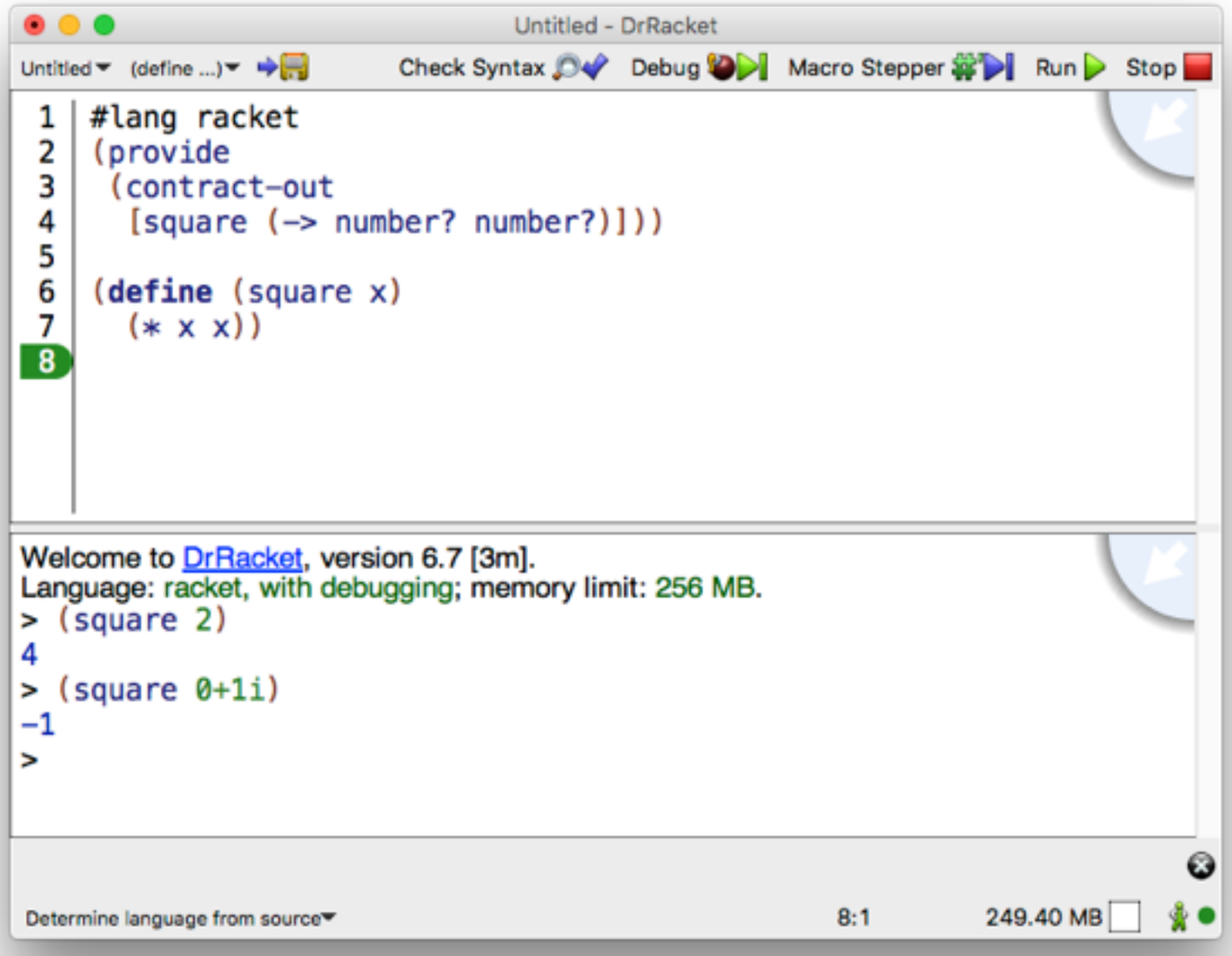
- Syntactic: structured, tiled, internationalized, etc.
- Semantic: “modes”, optimization levels, etc.
- Editor support: different views, user preferences, etc.

Examples of context orientation in languages

Context: application domain

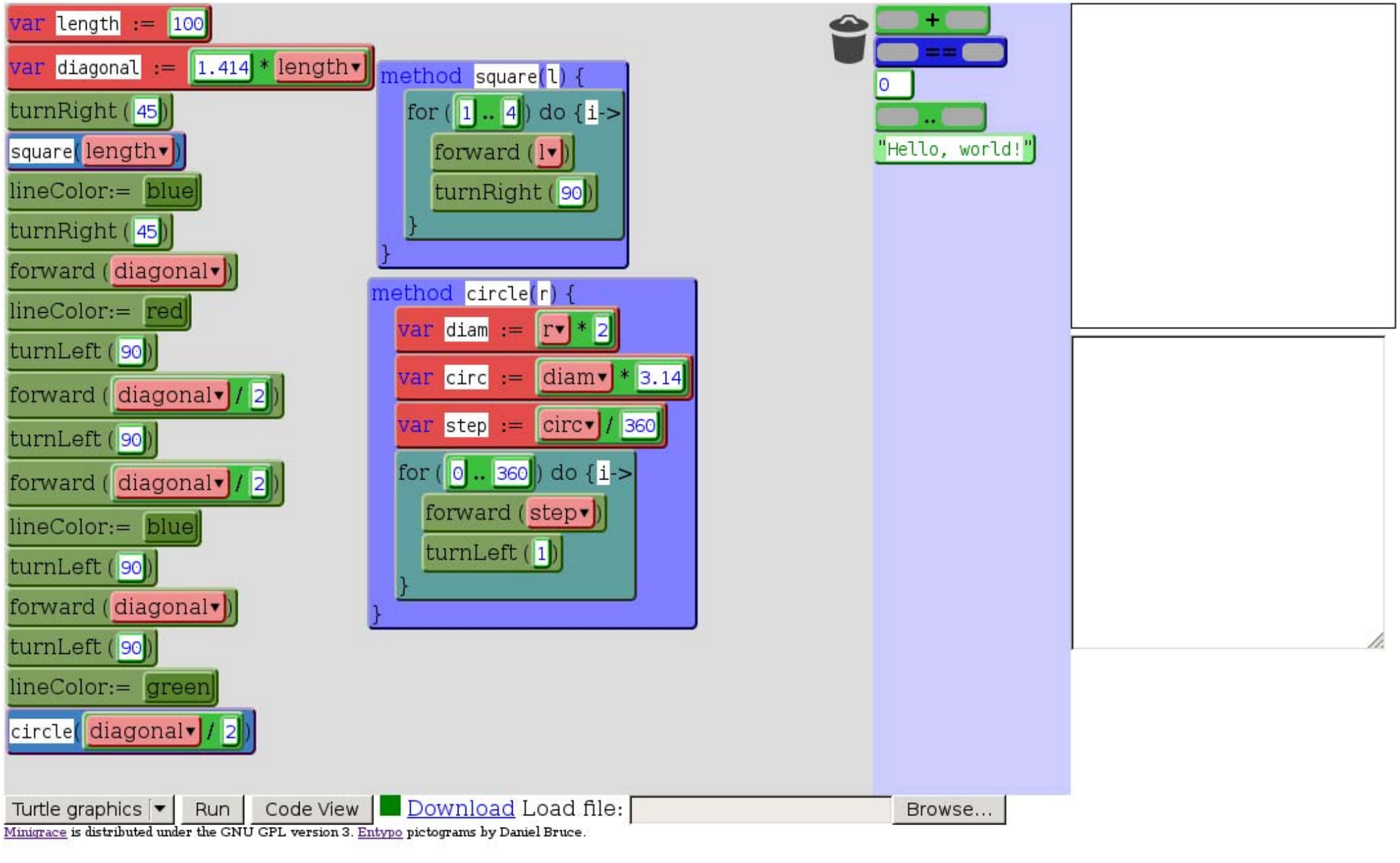


Context: proficiency

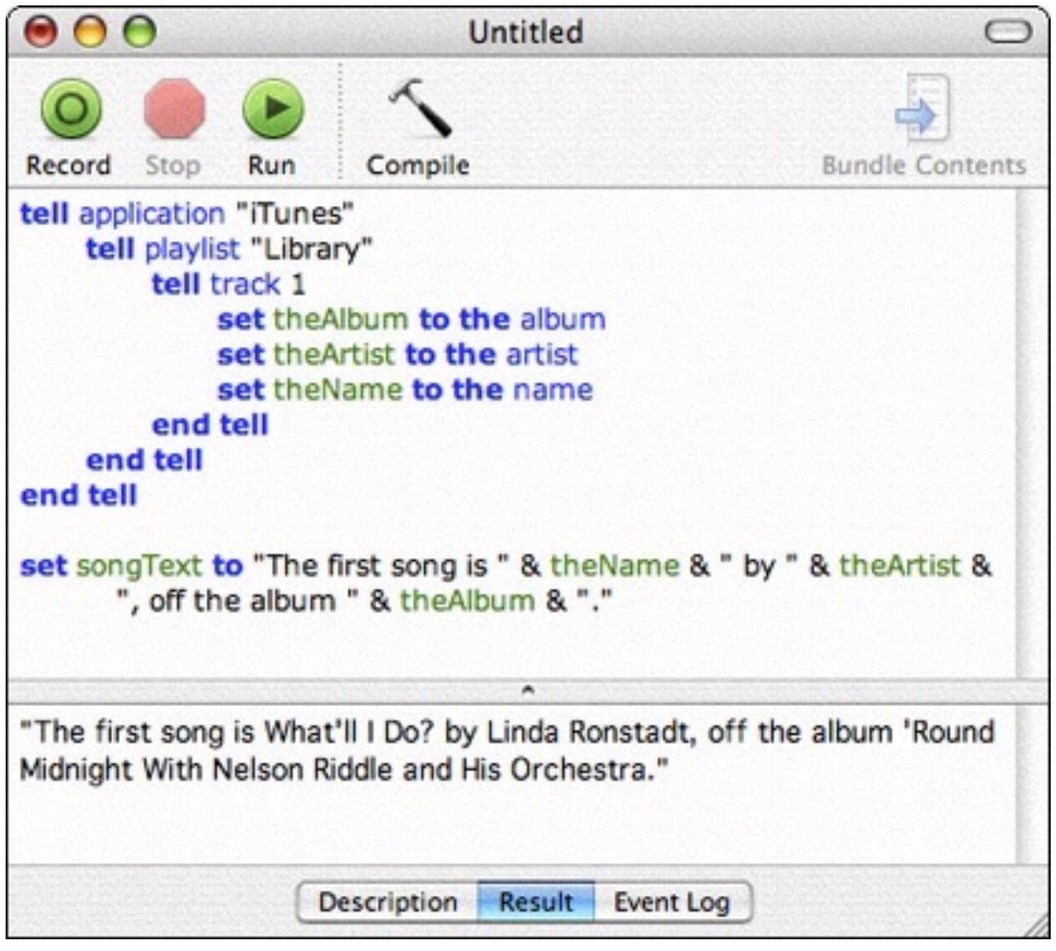


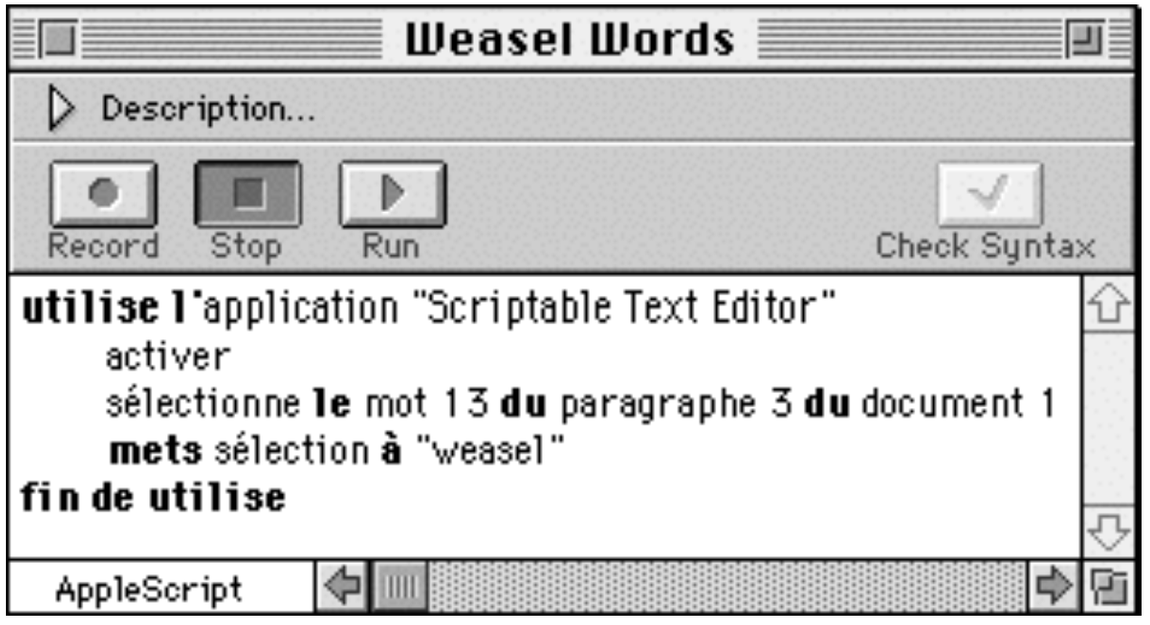
Context: proficiency

Tiled Grace



Context: locale





AppleScript

Context: locale+audience

```
De begindatum premieplicht ZVW van een IB belastingplichtige moet gesteld worden op leeg indien
zijn indicatie geheel jaar geen ZVW plicht (ja) gevuld is.  /* Zorgverzekeringswet, art 2 */

//begindatum premieplicht ZVW 02
De begindatum premieplicht ZVW van een IB belastingplichtige moet gesteld worden op zijn begindatu
indien hij aan alle volgende voorwaarden voldoet:
- zijn indicatie geheel jaar geen ZVW plicht (ja) is leeg
- zijn begindatum afwijkende periodeplicht ZVW is gevuld.   /* Zorgverzekeringswet, art 2 */

//begindatum premieplicht ZVW 03
De begindatum premieplicht ZVW van een IB belastingplichtige moet gesteld worden op $BEGINDATUM BE
indien hij aan alle volgende voorwaarden voldoet:
- zijn indicatie geheel jaar geen ZVW plicht (ja) is leeg
- zijn begindatum afwijkende periodeplicht ZVW is leeg. /* Zorgverzekeringswet, art 2 */

//einddatum premieplicht ZVW 01
De einddatum premieplicht ZVW van een IB belastingplichtige moet gesteld worden op leeg
indien zijn indicatie geheel jaar geen ZVW plicht (ja) gevuld is.   /* Zorgverzekeringswet, art 2

//einddatum premieplicht ZVW 02
De einddatum premieplicht ZVW van een IB belastingplichtige moet gesteld worden op zijn einddatum
indien hij aan alle volgende voorwaarden voldoet:
- zijn indicatie geheel jaar geen ZVW plicht (ja) is leeg
- zijn einddatum afwijkende periodeplicht ZVW is gevuld.    /* Zorgverzekeringswet, art 2 */

//einddatum premieplicht ZVW 03
```

Context: execution phase

TwoStones: Immediate Mode with Immutable Data

Code

```
1  def app(m)
2    div {
3      button {
4        out "+"
5        on click m.count = m.count + 1
6      }
7      out m.count
8      button {
9        out "-"
10       on click m.count = m.count - 1
11     }
12   }
13
14 model
15  {count: 0}
```

View

+ 0 -

Model

- count: 0

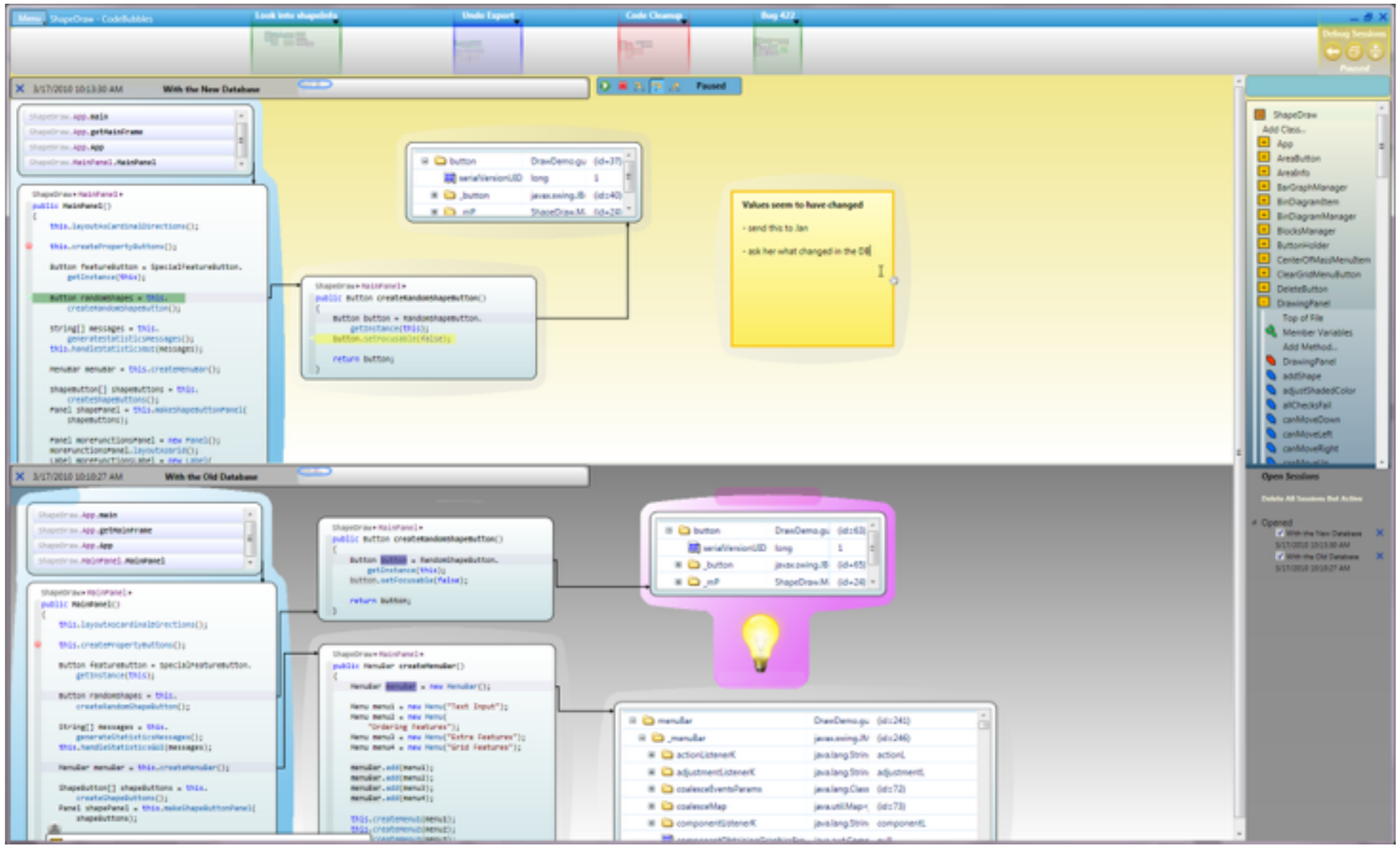
History

Prev Next Clear

. = {count: 0}...
.count = 1
.count = 2
.count = 1
.count = 0

van der Storm, *Immediate Mode with Immutable Data*, ProWeb'17

Context: dev task



Context: privacy policy

Policy agnostic programming with

Jeeves 

by @jeanqasaur http://projects.csail.mit.edu/jeeves/

Context: activity

```
table grades = #  A / B   / C    / D                  / E
                1:   | Lab | Exam | Avg                | Grade
                2:   | 7   | 7    | = (B2 + B3) / 2 | = round(D2)
                3:   | 3   | 7    | = (B3 + C3) / 2 | = round(D3)
                4:   | 9   | 10   | = (B4 + C4) / 2 | = round(D4).


view grades = #  A / B   / C    / D   / E
                1:   | Lab | Exam | Avg | Grade
                2:   | 7   | 7    | 5.  | 5.
                3:   | 3   | 7    | 5.  | 5.
                4:   | 9   | 10   | 9.5 | 10..

test grades E2 * 2 == B2 + C2 expected 14., got 10.

repl for grades
> A2 + B2.
=> 7.0
>B2 + B2.
=> 7.0
>
```

Context: activity

```
⊕
cell(function avg(sheet, lab, exam) {
  return (lab + exam) / 2;
});
```

```
cell(function classAvg(sheet) {
  return avg('avg', sheet);
});

var grades = sheet([
 {classAvg: 29.375},
 {student: 'James', lab: 93, exam: 9, avg: 51},
 {student: 'Sean', lab: 8.5, exam: 7, avg: 7.75}
]);
```

Language Context

- Contextual variants of syntax, semantics, editor, IDE support, etc.
- Outer influences inner: use, locale, task, policies, preference, etc.
- How to engineer such languages?
- What are the reusable “principled” techniques?

Let’s zoom in

- COP interpreters in Rascal
- Customization through “extreme” modularity
- Context propagation through interpreters using Object Algebras

Context-oriented programming an interpreter

- Simple functional language
- Interpreter gets another activated-layers argument
- Example case: “-Dassert” layer
- Activation: top-level or through syntax

rascal-mpl.org

Rascal: a meta programming language

- Functional programming with curly braces
- Runs on the JVM

- Command line REPL + Eclipse-based IDE
- Source: https://github.com/usethesource/rascal
- Download: http://www.rascal-mpl.org

AWK

ANTLR

grep

SQL

etc.

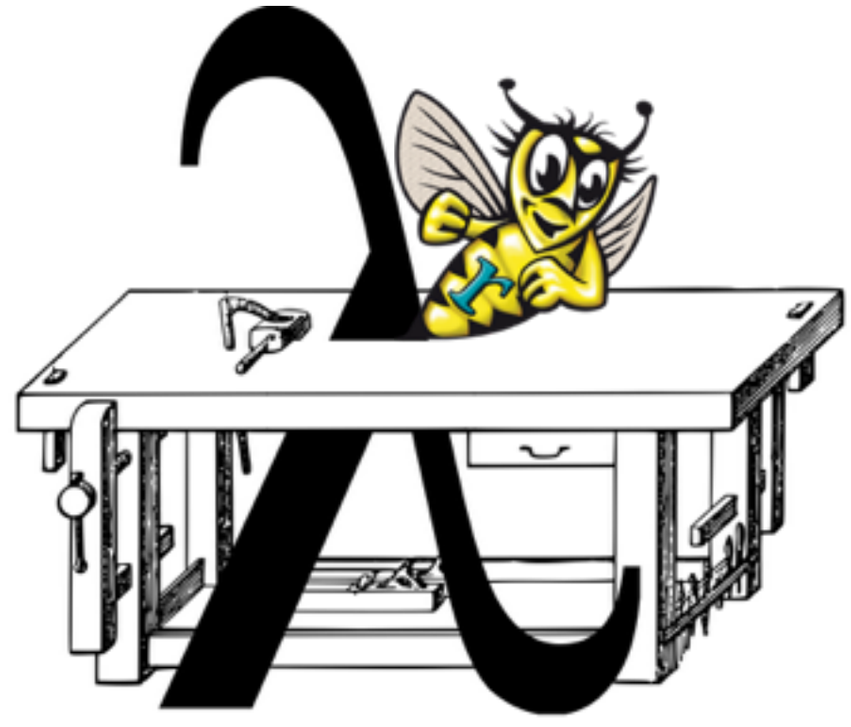
Rascal

http://www.rascal-mpl.org

http://usethesource.io/

Rascal as a language workbench

- Language workbench
  = “compiler compiler”
  + IDE support



```
fun fac(n) =
  assert n ⩾ 0;
  if n ⩾ 1 then
     n * fac(n - 1)
  else
     1
  fi

fac(10)
```

```
fun fac(n) =
  assert n ⩾ 0;
  if n ⩾ 1 then
     n * fac(n - 1)
  else
     1
  fi

asserting {
  fac(10)
}
```

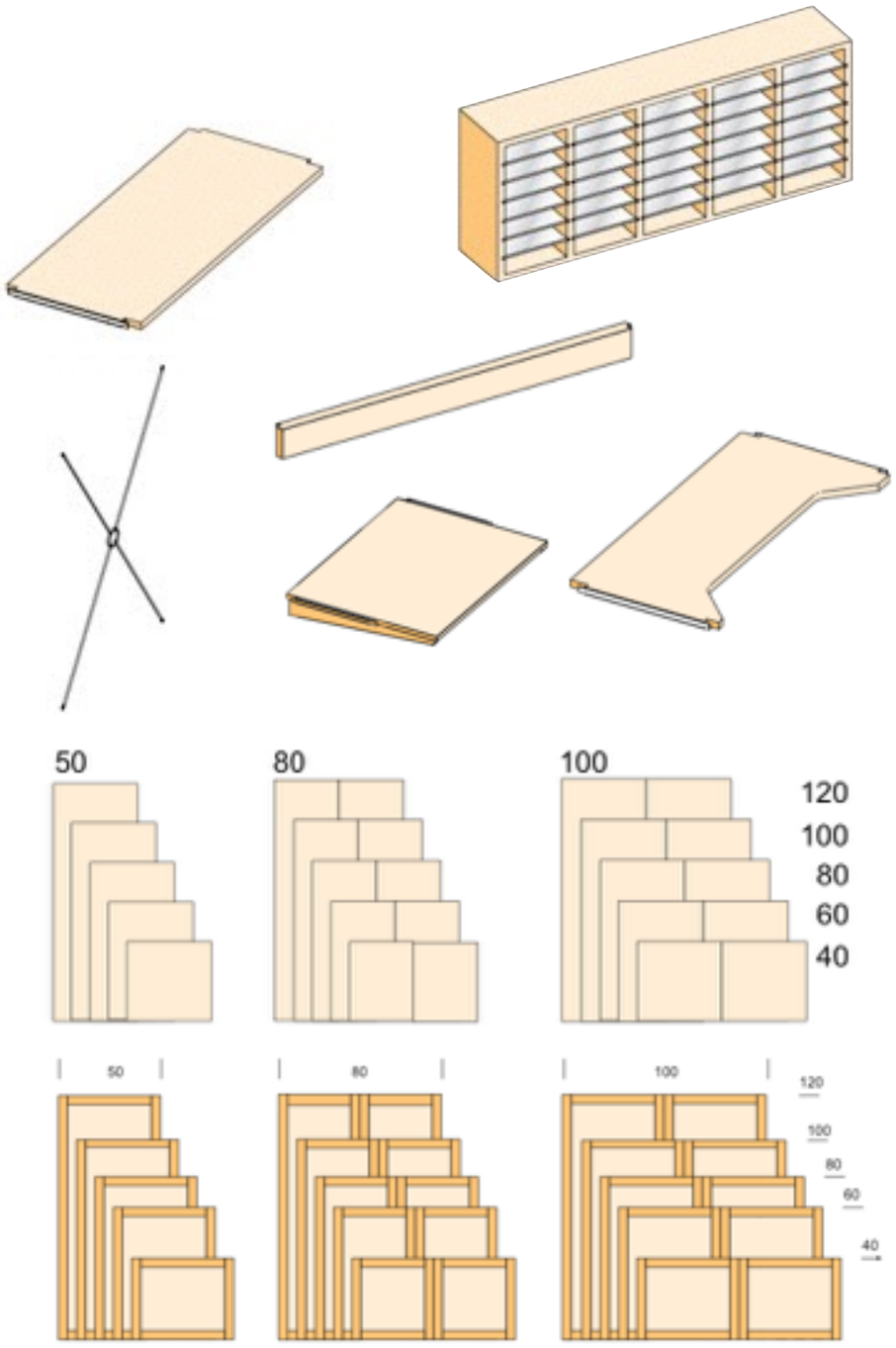
Live coding…

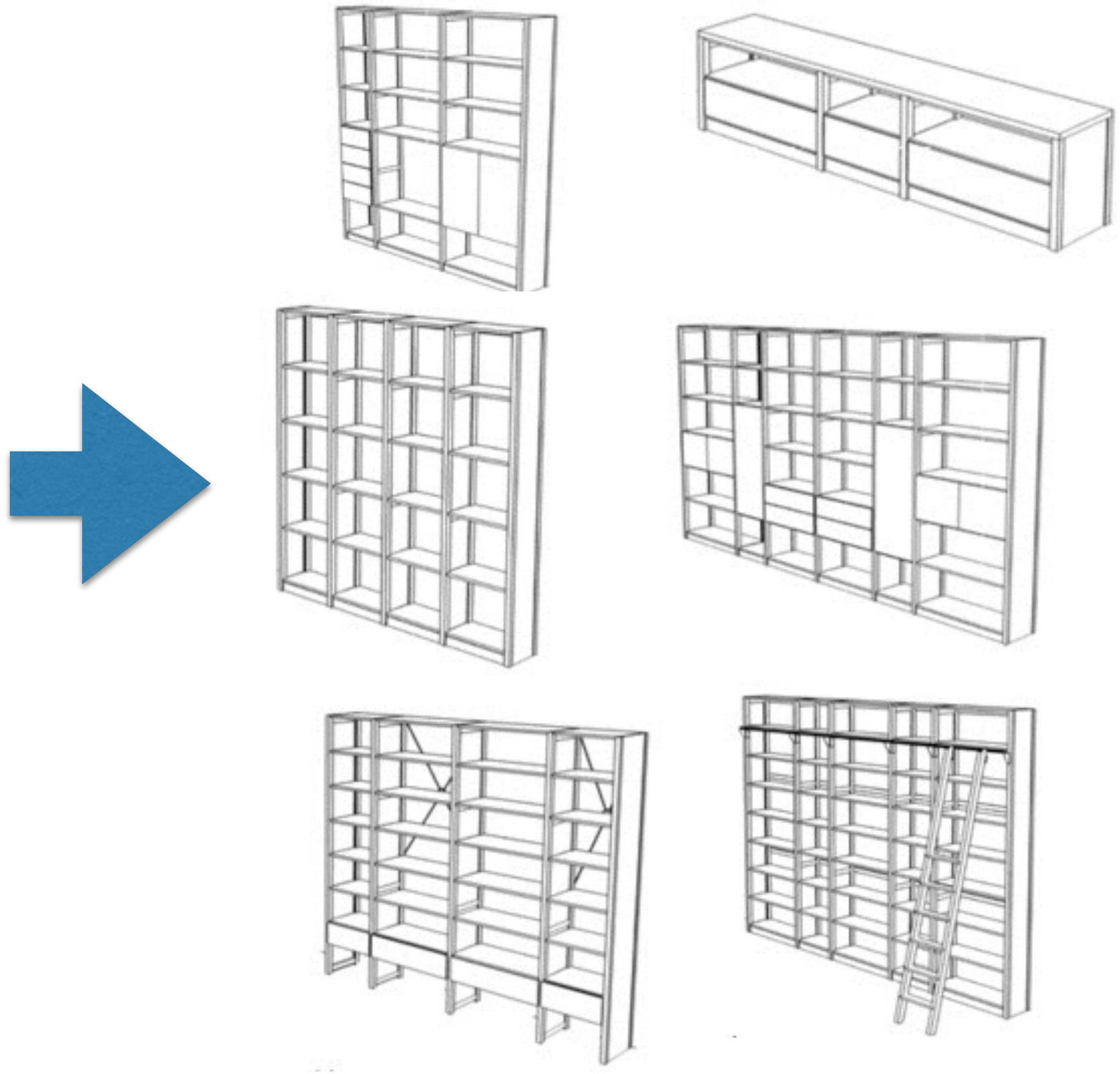
COP interpreter

- Context propagation through extra dispatch parameter
- If the “layer” is enabled, evaluate assert
- User can enable it “dynamically” through “asserting”

Language customization through composition

Towards Lundia Languages





Object Algebras

- A design pattern for defining interpretations

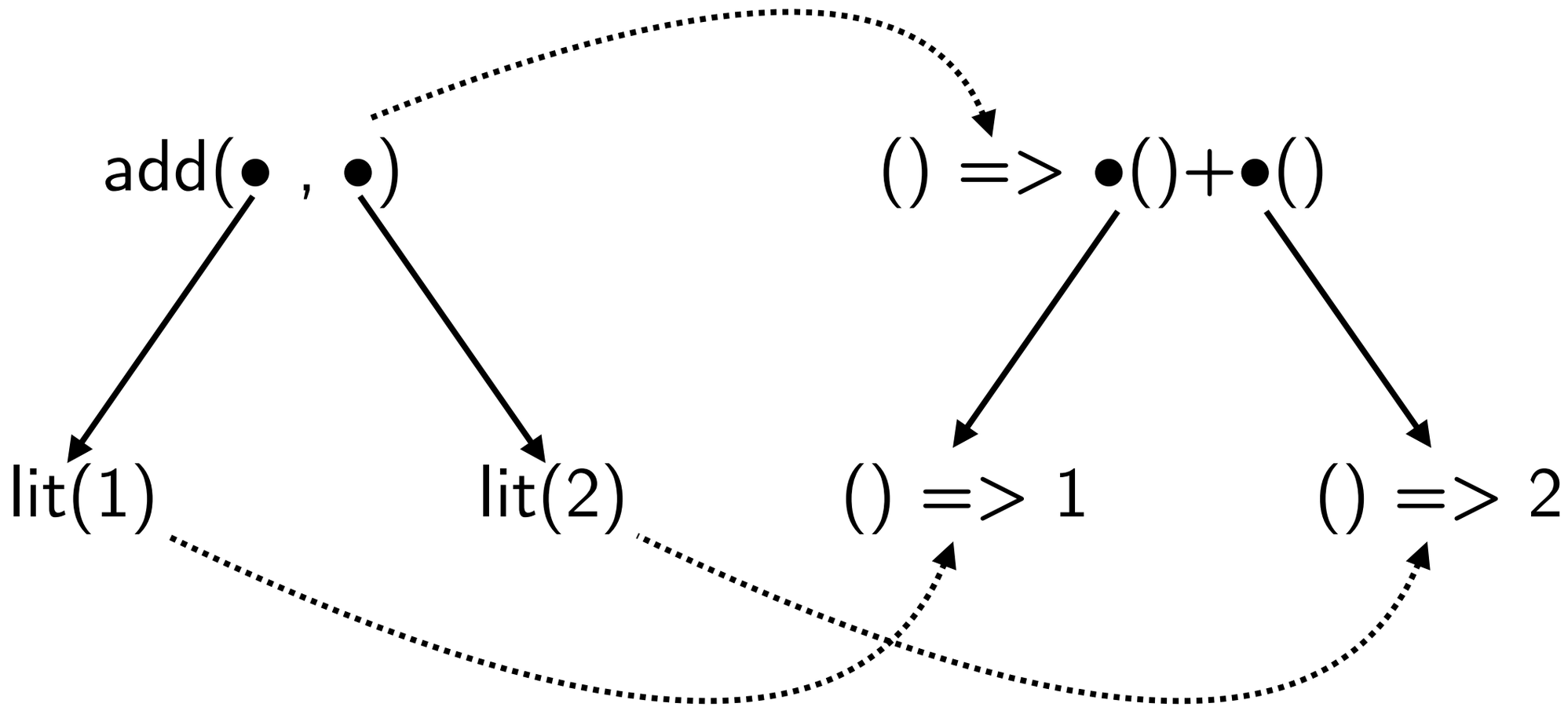
- Supports extreme modularity:
  - extension of language with new syntax
  - addition of another interpretation
- In both directions: **type safe**, **modular**, **separate compilation**.

Oliveira, Cook, ECOOP'12

```
interface ExpAlg<E> {
  E Add(E l, E r);
  E Lit(int n);
}
```

```
interface IEval {
  int eval();
}

interface EvalExp extends ExpAlg<IEval> {
  default IEval Add(IEval l, IEval r) {
    return () → l.eval() + r.eval();
  }
  default IEval Lit(int n) {
    return () → n;
  }
}
```

“1 + 2”



```
interface PrintExp extends ExpAlg<String> {
  default String Add(String l, String r) {
    return l + " + " + r;
  }
  default String Lit(int n) {
    return "" + n;
  }
}
```

```
interface SubAlg<E> extends ExpAlg<E> {
  E Sub(E l, E r);
}

interface EvalSub extends SubAlg<IEval> {
  default IEval Sub(IEval l, IEval r) {
    return () → l.eval() - r.eval();
  }
}

interface PrintSub extends SubAlg<String> {
  default String Sub(String l, String r) {
    return l + " - " + r;
  }
}
```

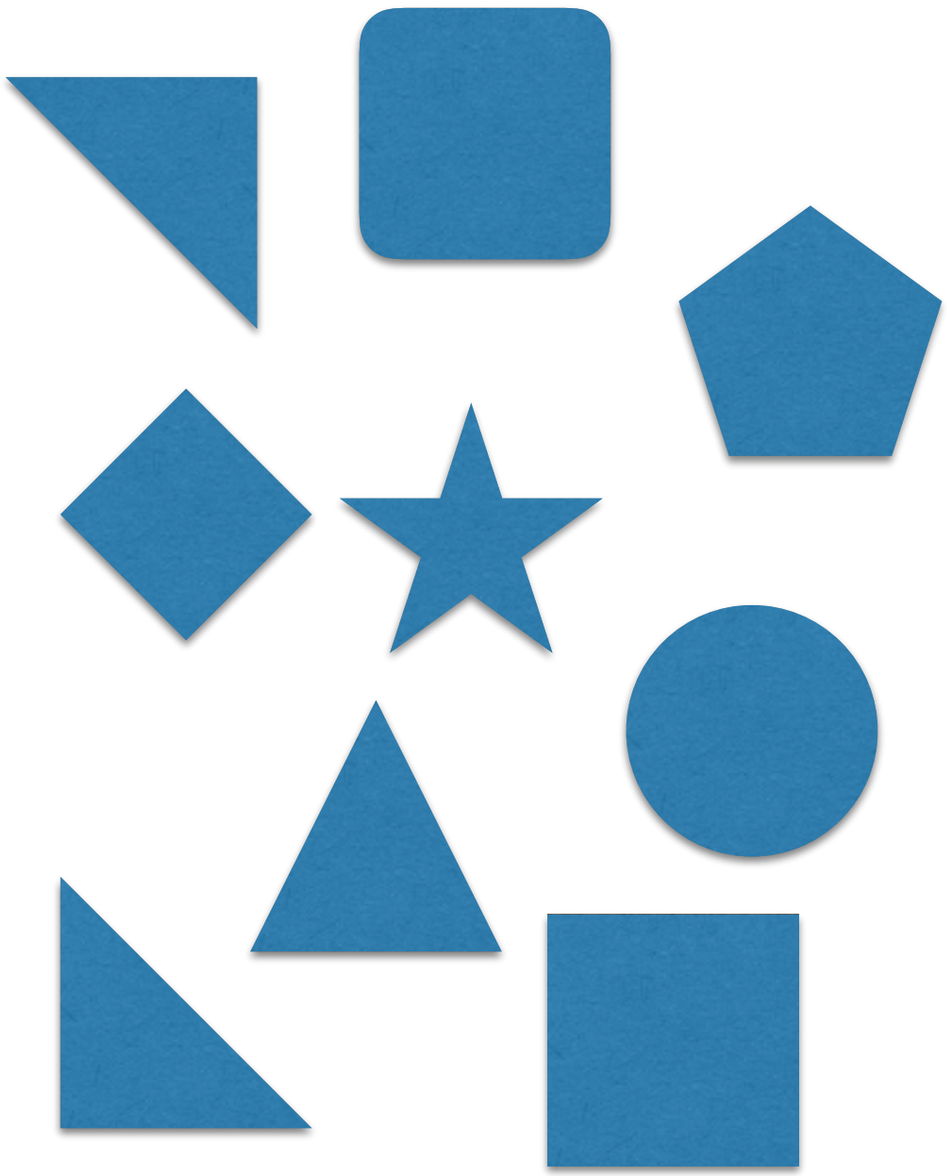
Implicit context propagation

- Interpreters can be extended, but extensions may have **incompatible** signatures
  - Arithmetic expressions: ***() → Value*** (“IEval")
  - Binding constructs: ***Env → Value***
- Solution: generate another interpretation to lift the “smaller” one, and propagate the additional argument (e.g., **Env**).

Inostroza, vd Storm, *COMLAN*, Volume 48, June 2017, pp 39–67

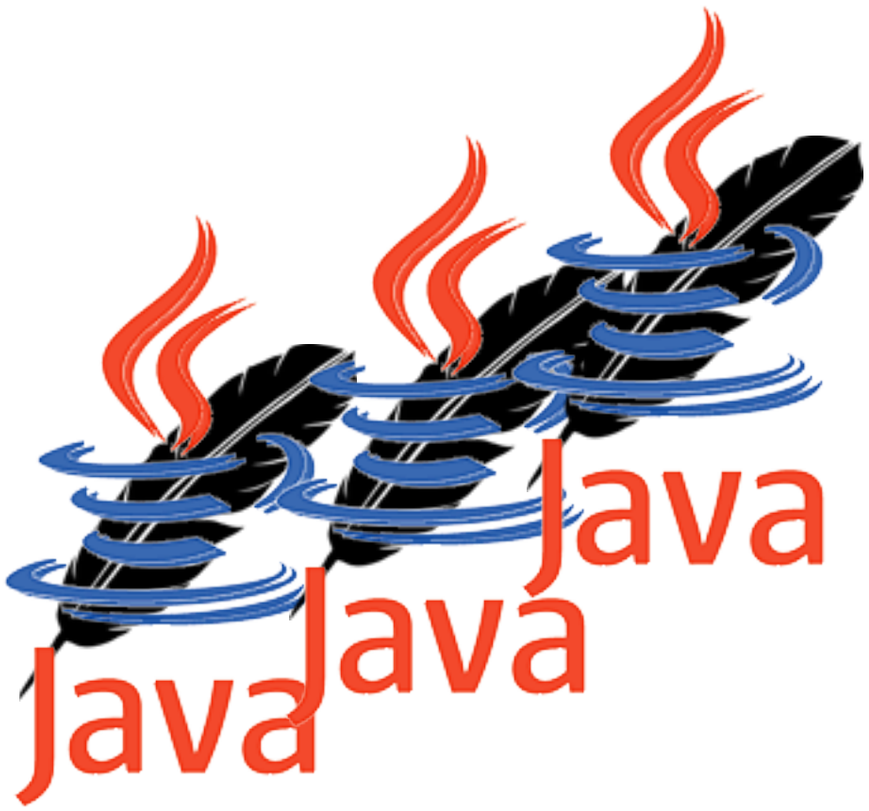
```
interface IEvalEnv {
   int eval(Env env);
}

interface EvalExpEnv extends ExpAlg<IEvalEnv> {
   ExpAlg<IEval> base();

   default IEvalEnv Add(IEvalEnv l, IEvalEnv r) {
      return env → base().Add(() → l.eval(env),
                              () → r.eval(env)).eval();
   }
   default IEvalEnv Lit(int n) {
      return env → base().Lit(n).eval();
   }
}
```

Extreme modularity



9 semantic components

127 variants of
Featherweight Java



. . .

Featherweight Java

	Syntax	Signature
Field access	`e.f`	`CT => Obj`
Object Creation	**`new`** `C(e,…)`	`(CT, Sto) => Obj`
Casting	`(C) e`	`CT => Obj`
Variables	`x`	`(Obj, CT, Env, Sto) => Obj`
Method Call	`e.m(e, …)`	`(Obj, CT, Env) => Obj`
Sequencing	`e ; e`	`() => Obj`
Field Assignment	`e.f = e`	`(CT, Sto) => Obj`

What this enables

- Language components without anticipation of context
- Flexible customization/configuration of languages.
- E.g. combine expressions that have side-effects with expressions without side-effects in the same language, depending on syntactic context.

```
interface AssertAlg<E> {
  E Assert(E e);
}

interface IEvalCtx {
  int eval(boolean asserting);
}

interface EvalAssert extends AssertAlg<IEvalCtx> {
  @Override
  default IEvalCtx Assert(IEvalCtx e) {
    return a → {
      if (a && e.eval(a) == 0)
        throw new AssertionError();
      return 0;
    };
  }
}
```

```
interface AssertingAlg<E> {
  E Asserting(E e);
}

interface EvalAsserting extends AssertingAlg<IEvalCtx> {
  @Override
  default IEvalCtx Asserting(IEvalCtx e) {
    return a → e.eval(true);
  }
}
```

Related techniques

- Effect handlers (cf. Eff, Koka)

- Scala Implicits (Martin's CurryOn keynote)

Context propagation with Object Algebras

- Extreme modularity => cherry picking language features
- Context propagation for semantic variation
- Future (?): dynamic selection of language features
- Towards personalization of languages!

Context-oriented Language Engineering

- Context = important for software languages :)
- Need to make languages “aware” of context
- Many examples of context-dependence
- Context propagation ~ COP
- Only tip of the ice berg…

Lots of future directions

- Adaptive syntax
- Dynamic loading of language extensions
- IDEs learning programmer preferences
- …