Adding an Extensible Backend to PQL/Java

Christoph Reichenbach
R^2 Software & Systeme UG

19 July 2017 / WG 2.11

based on Hilmar Ackermann, Christoph Reichenbach, Christian Müller, Yannis Smaragdakis. ‘A Backend Extension Mechanism for PQL/Java with Free Run-Time Optimisation’, in CC’15.

PQL/Java

- **P**arallel **Q**uery **L**anguage
- Embedded Declarative DSL as Java extension
- First-Order Logic-style queries over Java containers
- Automatic Parallelisation:
  - *Guaranteed* parallelisation (with 'right' types)

```
Set<Point> result =
    query(Set.contains(Point e)):
          s1.contains(e)
       && s2.contains(e)
       && e.x > 0;
```

PQL/Java Operators

- !, ~, +, -, ..., ?:, ==, **instanceof**, &&, ||, ->
- **forall**, **exists**
- Java expressions as constants
- m[k], m.get(k), c.length, c.size(), s.contains(e)
- Container construction:
  - **query**(Set.contains(int x)): ...
  - **query**(Array[x] == float f): ...
  - **query**(Map.get(String s) == int i [**default** v]): ...
- **reduce**(sumInt) int x [**over** y]: ...

PQL/Java Operators

- !, ~, +, -, ..., ?:, ==, **instanceof**, &&, ||, ->
- **forall**, **exists**
- Java expressions as constants
- m[k], m.get(k), c.length, c.size(), s.contains(e)
- Container construction:
  - **query**(Set.contains(int x)): ...
  - **query**(Array[x] == float f): ...
  - **query**(Map.get(String s) == int i [**default** v]): ...
- **reduce**(sumInt) int x [**over** y]: ...

That's it — what if we need more?

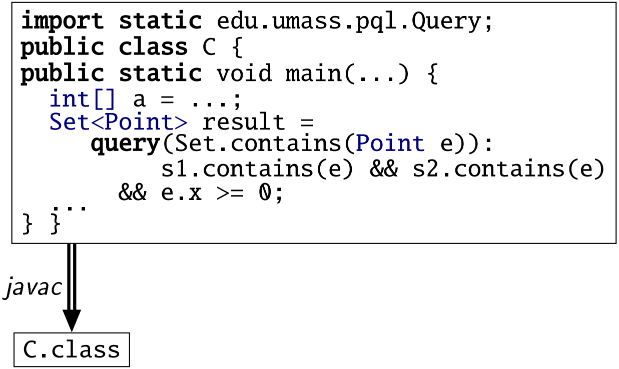
Making PQL Extensible

- TODO:
  - Extend syntax
  - Import existing Java
  - Extend analyses / optimisations
  - Extend code generation

Making PQL Extensible

- TODO:
  - Extend syntax (*Future work*)
  - Import existing Java (*Future work*)
  - **Extend analyses / optimisations**
  - **Extend code generation**
- Sort-of WIP

PQL Translation (Static Backend)

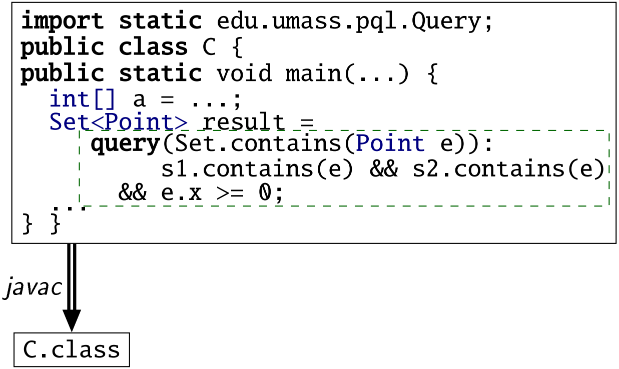


```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
  int[] a = ...;
  Set<Point> result =
     query(Set.contains(Point e)):
          s1.contains(e) && s2.contains(e)
       && e.x >= 0;
  ...
} }
```

javac

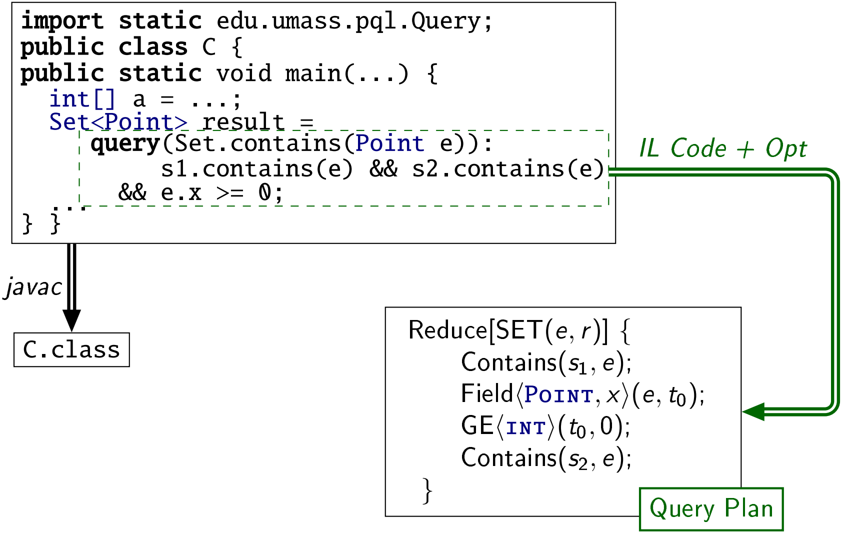
`C.class`

PQL Translation (Static Backend)



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
   int[] a = ...;
   Set<Point> result =
      query(Set.contains(Point e)):
             s1.contains(e) && s2.contains(e)
         && e.x >= 0;
   ...
} }
```

PQL Translation (Static Backend)



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
   int[] a = ...;
   Set<Point> result =
     query(Set.contains(Point e)):
          s1.contains(e) && s2.contains(e)
       && e.x >= 0;
   ...
} }
```

javac

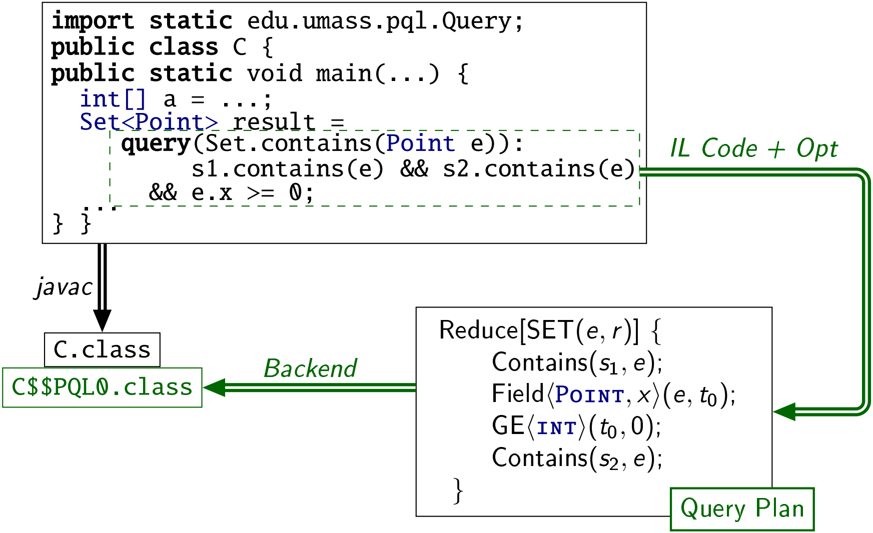
C.class

IL Code + Opt

```
Reduce[SET($e, r$)] {
      Contains($s_1, e$);
      Field$\langle \textsc{Point}, x \rangle(e, t_0)$;
      GE$\langle \textsc{int} \rangle(t_0, 0)$;
      Contains($s_2, e$);
  }
```

Query Plan

PQL Translation (Static Backend)



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
   int[] a = ...;
   Set<Point> result =
      query(Set.contains(Point e)):
           s1.contains(e) && s2.contains(e)
        && e.x >= 0;
   ...
} }
```

IL Code + Opt

javac

`C.class`

`C$$PQL0.class`

Backend

Reduce[SET(e, r)] {
    Contains(s_1, e);
    Field$\langle$POINT$, x\rangle(e, t_0)$;
    GE$\langle$INT$\rangle(t_0, 0)$;
    Contains(s_2, e);
}

Query Plan

Access Modes

$\mathsf{Contains}(s_1, e)$
$\mathsf{Field}\langle\textsc{Point}, x\rangle(e, t_0)$
$\mathsf{GE}\langle\textsc{int}\rangle(t_0, 0)$
$\mathsf{Contains}(s_2, e)$

```
foreach $e \in s_1$:
```

Access Modes

$\mathsf{Contains}(s_1, e)$
$\mathsf{Field}\langle \textsc{Point}, x\rangle(e, t_0)$
$\mathsf{GE}\langle \textsc{int}\rangle(t_0, 0)$
$\mathsf{Contains}(s_2, e)$

```
foreach $e \in s_1$:
  $t_0 := e$.x
```

Access Modes

$\text{Contains}(s_1, e)$
$\text{Field}\langle \textsc{Point}, x \rangle(e, t_0)$
$\text{GE}\langle \textsc{int} \rangle(t_0, 0)$
$\text{Contains}(s_2, e)$

```
foreach $e \in s_1$:
  $t_0 := e$.x
  if $t_0 \geq 0$:
```

Access Modes

$\text{Contains}(s_1, e)$
$\text{Field}\langle\textsc{Point}, x\rangle(e, t_0)$
$\text{GE}\langle\textsc{Int}\rangle(t_0, 0)$
$\text{Contains}(s_2, e)$

```
foreach $e \in s_1$:
  $t_0 := e$.x
  if $t_0 \geq 0$:
    if $e \in s_2$:
      (insert $e$ into result set)
```

Access Modes

$\text{Contains}(s_1, e)$
$\text{Field}\langle \texttt{POINT}, x\rangle(e, t_0)$
$\text{GE}\langle \texttt{INT}\rangle(t_0, 0)$
$\text{Contains}(s_2, e)$

```
foreach $e \in s_1$:
  $t_0 := e$.x
  if $t_0 \geq 0$:
    if $e \in s_2$:
      (insert $e$ into result set)
```

Access Modes

$\text{Contains}(s_1, e)$
$\text{Field}\langle\textbf{Point}, x\rangle(e, t_0)$
$\text{GE}\langle\textbf{int}\rangle(t_0, 0)$
$\text{Contains}(s_2, e)$

```
foreach $e \in s_1$:
  $t_0$ := $e$.x
  if $t_0 \geq 0$:
    if $e \in s_2$:
      (insert $e$ into result set)
```

- $\text{Contains}(s_1, e^w)$: *Write* to x (iterate over all values)
- $\text{Contains}(s_1, e^r)$: *Read* x (contains-check)

Access Modes

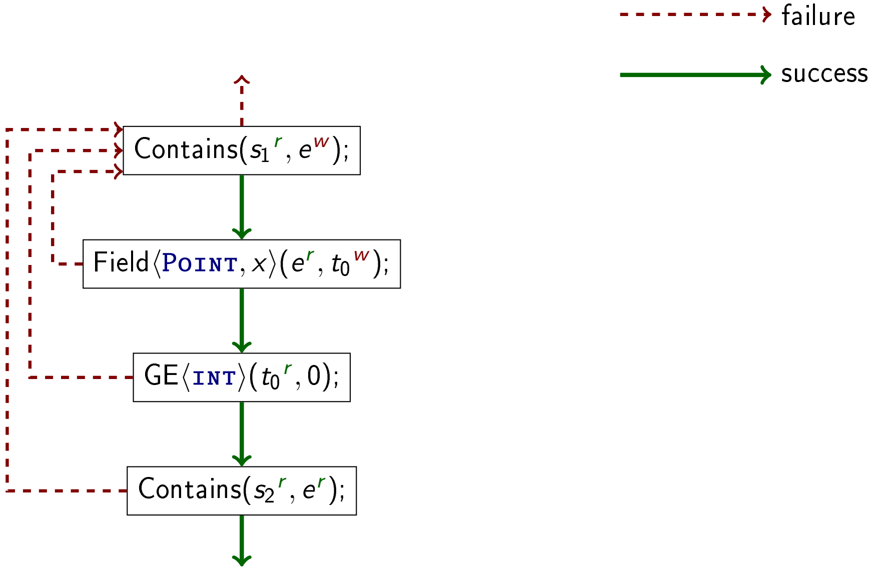
$\text{Contains}({s_1}^r, e^w)$
$\text{Field}\langle\textbf{Point}, x\rangle(e^r, {t_0}^w)$
$\text{GE}\langle\textbf{Int}\rangle({t_0}^r, 0)$
$\text{Contains}({s_2}^r, e^r)$

```
foreach $e \in s_1$:
  $t_0 := e.\mathtt{x}$
  if $t_0 \geq 0$:
    if $e \in s_2$:
      (insert $e$ into result set)
```

- $\text{Contains}(s_1, e^w)$: *Write* to x (iterate over all values)
- $\text{Contains}(s_1, e^r)$: *Read* x (contains-check)

Backend must utilise access mode information

Control Flow



failure

success

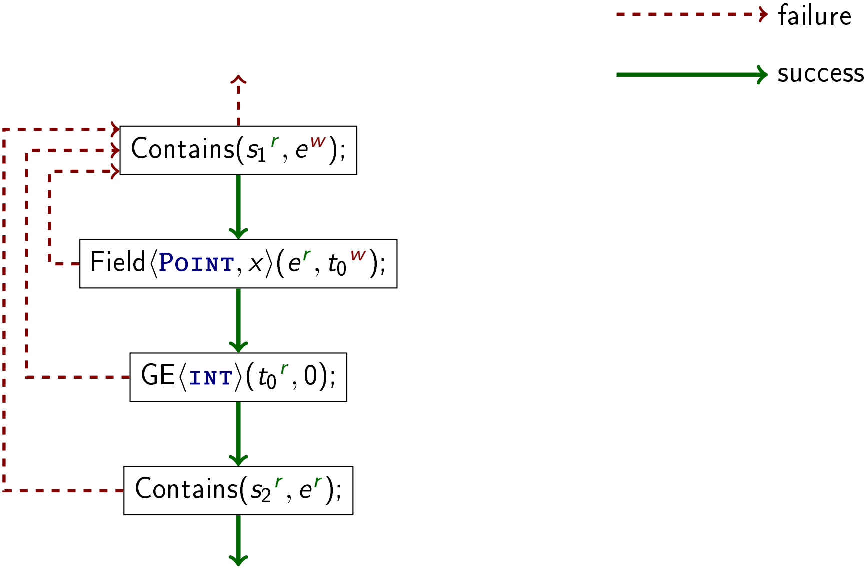
$\text{Contains}(s_1{}^r, e^w);$

$\text{Field}\langle\texttt{POINT}, x\rangle(e^r, t_0{}^w);$

$\text{GE}\langle\texttt{INT}\rangle(t_0{}^r, 0);$

$\text{Contains}(s_2{}^r, e^r);$

Control Flow



Contains$(s_1{}^r, e^w)$;

Field$\langle$**Point**$, x\rangle(e^r, t_0{}^w)$;

GE$\langle$**int**$\rangle(t_0{}^r, 0)$;

Contains$(s_2{}^r, e^r)$;

Backend must explicitly support success/failure

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference

$$\boxed{\mathrm{GE}\langle \texttt{INT} \rangle ({t_0}^{r}, 0);}$$

```
@accessModes{rr}
ge(val1, val2) {
local:
    if ( @type{int} val1 >= val2) proceed;
    else abort;
}
```

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference

Supported access modes

```
@accessModes{rr}
ge(val1, val2) {
local:
    if ( @type{int} val1 >= val2) proceed;
    else abort;
}
```

$\boxed{\mathrm{GE}\langle\mathtt{INT}\rangle({t_0}^{r}, 0);}$

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference

Supported access modes

```
@accessModes{rr}
ge(val1, val2) {
local:
    if ( @type{int} val1 >= val2) proceed;
    else abort;
}
```

Entry point label

$$\boxed{\mathrm{GE}\langle \texttt{INT} \rangle ({t_0}^{r}, 0);}$$

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference

Supported access modes

$\boxed{\mathrm{GE}\langle\mathbf{INT}\rangle({t_0}^{r}, 0);}$

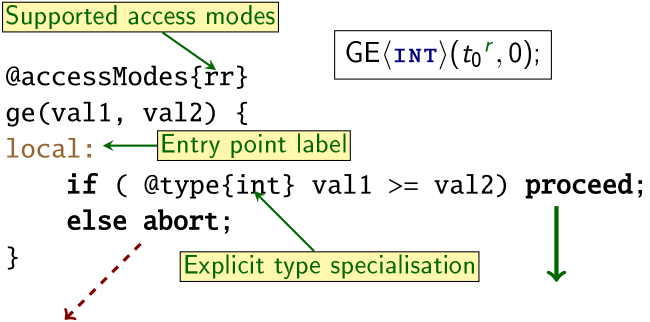
```
@accessModes{rr}
ge(val1, val2) {
local:
    if ( @type{int} val1 >= val2) proceed;
    else abort;
}
```

Entry point label

Explicit type specialisation

Extending PQL with PQL-ESL

- PQL-ESL: PQL **E**xtension **S**pecification **L**anguage
- Describes semantics of PQL Intermediate Language operators
- Simplified Java-like syntax with some extensions and type inference



Supported access modes

$\mathrm{GE}\langle\texttt{INT}\rangle({t_0}^{r}, 0);$

```
@accessModes{rr}
ge(val1, val2) {
local:
    if ( @type{int} val1 >= val2) proceed;
    else abort;
}
```

Entry point label

Explicit type specialisation

Iteration in PQL-ESL

$$\boxed{\text{Contains}(s_1{}^{r}, e^{w});}$$

```
@accessModes{rr, rw}
contains(set, element)
{
local:
    if (isMode( (set,element), (rw) )) {
        it = set.getIterator();
iteration:
        hasNext = it.hasNext();
        if (hasNext == 0)
            abort;
        element = it.next();
        proceed;
    }




}
```

Iteration in PQL-ESL

```
@accessModes{rr, rw}
contains(set, element)
{
local:
    if (isMode( (set,element), (rw) )) {
        it = set.getIterator();
iteration:
        hasNext = it.hasNext();
        if (hasNext == 0)
            abort;
        element = it.next();
        proceed;
    }




}
```

Explicit access mode check

$\text{Contains}({s_1}^{r}, e^{w});$

Iteration in PQL-ESL

```
@accessModes{rr, rw}
contains(set, element)
{
local:
    if (isMode( (set,element), (rw) )) {
        it = set.getIterator();
iteration:
        hasNext = it.hasNext();
        if (hasNext == 0)
            abort;
        element = it.next();
        proceed;
    }





}
```

Explicit access mode check

Contains($s_1{}^r$, e^w);

Iteration entry point label

Iteration in PQL-ESL

```
@accessModes{rr, rw}
contains(set, element)
{
local:
    if (isMode( (set,element), (rw) )) {
        it = set.getIterator();
iteration:
        hasNext = it.hasNext();
        if (hasNext == 0)
            abort;
        element = it.next();
        proceed;
    }








}
```

$\text{Contains}(s_1{}^r, e^w);$

Explicit access mode check

Iteration entry point label

Assignment to parameter: *reference semantics*

Iteration in PQL-ESL

```
@accessModes{rr, rw}
contains(set, element)
{
local:
    if (isMode( (set,element), (rw) )) {
        it = set.getIterator();
iteration:
        hasNext = it.hasNext();
        if (hasNext == 0)
            abort;
        element = it.next();
        proceed;
    }
    if (isMode( (set,element), (rr)
        tmpElement = set.contains(element);
        if (tmpElement == 0) abort;
        else proceed;
    }
}
```

Explicit access mode check

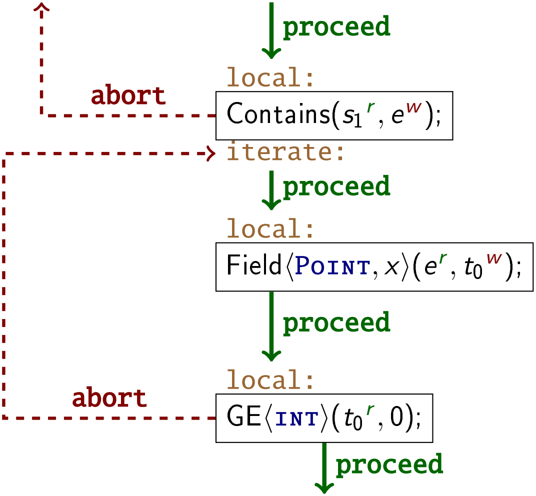
$\mathsf{Contains}(s_1{}^r, e^w);$

Iteration entry point label

Assignment to parameter: *reference semantics*

$\mathsf{Contains}(s_2{}^r, e^r);$

Control Flow



proceed

local:

Contains($s_1{}^r, e^w$);

abort

iterate:

proceed

local:

Field⟨Point, x⟩($e^r, t_0{}^w$);

proceed

local:

GE⟨int⟩($t_0{}^r$, 0);

abort

proceed

- **proceed** jumps to next local:
- **abort** jumps to most recent iterate:

Support for Linear Operators

- Linear operators are common:
  - arithmetic
  - bit operations
  - typical Java calls
- General pattern: $\mathit{result} = [\mathit{arg0.}]f(\mathit{args})$
- Example: $\mathrm{Add}\langle\mathtt{INT}\rangle(x^r, y^r, \mathit{result})$

Support for Linear Operators

- Linear operators are common:
  - arithmetic
  - bit operations
  - typical Java calls
- General pattern: $\mathit{result} = [\mathit{arg0}.] f(\mathit{args})$
- Example: $\mathrm{Add}\langle \mathtt{INT} \rangle (x^r, y^r, \mathit{result})$

```
tmp = f(args);
if (isMode( (result), (r) )) {
    if (result == tmp) proceed;
    else abort;
} else {
    result = tmp;
    proceed;
}
```

Support for Linear Operators

- Linear operators are common:
  - arithmetic
  - bit operations
  - typical Java calls
- General pattern: $\mathit{result} = [\mathit{arg0.}]f(\mathit{args})$
- Example: $\mathrm{Add}\langle\texttt{INT}\rangle(x^r, y^r, \mathit{result})$

```
tmp = f(args);
if (isMode( (result), (r) )) {
    if (result == tmp) proceed;
    else abort;
} else {
    result = tmp;
    proceed;
}
```

proceed on *result* ?= f(*args*)

Templates for Types and Operators

- Parameterise types and operators

Templates for Types and Operators

- Parameterise types and operators
- Currently uses textual substitution

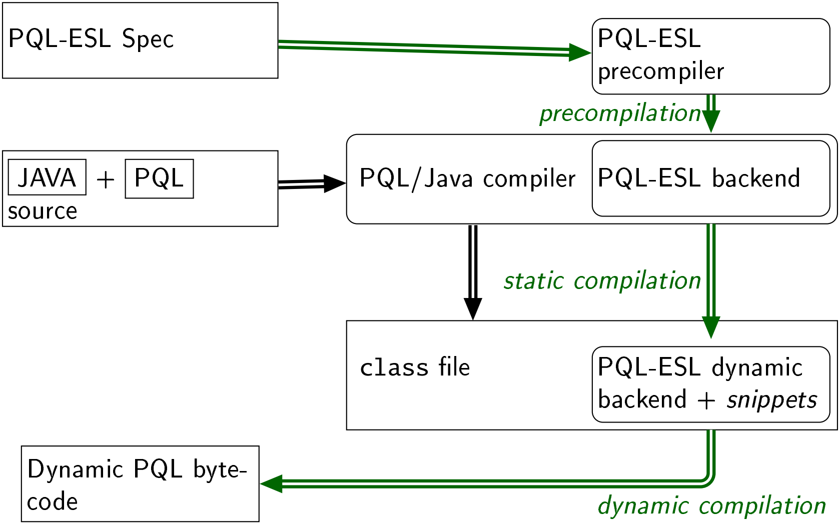
```
@generic{operator}{"<=", "<"}
@generic{type}{"int", "long", "double"}
@accessModes{rr}
lt_lte(val1, val2) {
local:
    if ( @type{#type#} val1 #operator# val2) proceed;
    else abort;
}
```

Templates for Types and Operators

- Parameterise types and operators
- Currently uses textual substitution
- Linked to IL operators in separate step

```
@generic{operator}{"<=", "<"}
@generic{type}{"int", "long", "double"}
@accessModes{rr}
lt_lte(val1, val2) {
local:
    if ( @type{#type#} val1 #operator# val2) proceed;
    else abort;
}
```

PQL Compilation Process



snippet: family of partially linked bytecode fragments, one per IL operator

Staging and Conditionals

1. **Pre**compilation
2. **Stat**ic compilation
3. **Dyn**amic compilation
4. **Exec**ution

Staging and Conditionals

1. **Pre**compilation
2. **Stat**ic compilation
3. **Dyn**amic compilation
4. **Exec**ution

- Conditions evaluated as early as possible, as late as necessary

		Pre	Stat	Dyn	Exec
`isMode`	access mode	+			
`instanceof`	dynamic type check		+	+	+

Staging and Conditionals

1. **Pre**compilation
2. **Stat**ic compilation
3. **Dyn**amic compilation
4. **Exec**ution

- Conditions evaluated as early as possible, as late as necessary

		Pre	**Stat**	**Dyn**	**Exec**
`isMode`	access mode	+			
`instanceof`	dynamic type check		+	+	+
`isConst(`x`)`	x is constant		+	+	

Staging and Conditionals

1. **Pre**compilation
2. **Stat**ic compilation
3. **Dyn**amic compilation
4. **Exec**ution

- Conditions evaluated as early as possible, as late as necessary

		Pre	**Stat**	**Dyn**	**Exec**
`isMode`	access mode	+			
`instanceof`	dynamic type check		+	+	+
`isConst(`x`)`	x is constant		+	+	
`isParallel()`	parallel execution			+	

- `isParallel` occurs in code
  ⇒ PQL assumes that the operator supports parallelisation

PQL-ESL, core feature summary

- Basic Java
- Genericity
- **proceed** and **abort**
- **proceed on**
- Access to static analysis
- Staging
- Other features:
  - Parallel execution
  - Custom Rewriting (WIP)
  - Cost model
  - Once-only Precomputation (`global:`)

Evaluation

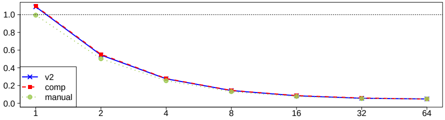
- Re-implemented PQL backend using PQL-ESL
- Added support for dynamic compilation
- Added extensions:
  - `sqrt`
  - modulo
  - `isPrime(`*n*`)` and `primesRange(`*min*`,` x`,` *max*`)`
  - Java 8 Streams
  - SQL
- Added rewriting:
  - `isPrime` + `range` ⇒ `primesRange`
  - PQL ⇒ DB access operators

  etc.

General Performance

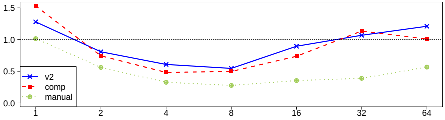
wordcount

normalized to 3330.720855ms



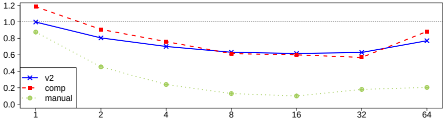
threegrep

normalized to 22.831667ms



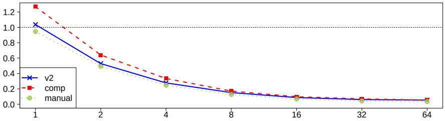
bonus

normalized to 250.991431ms

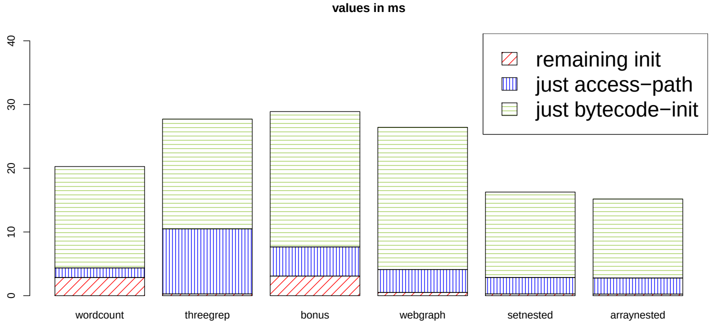


webgraph

normalized to 1634.369175ms

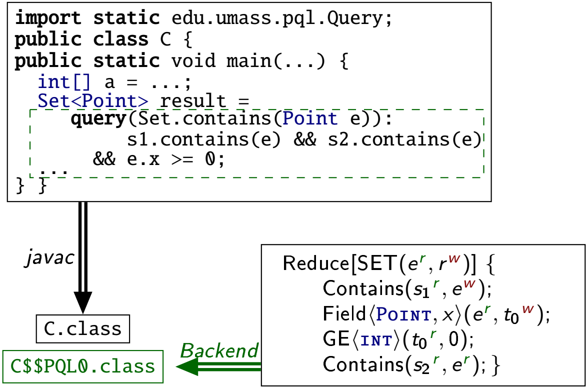


Overhead



Dynamic compilation comes at a price

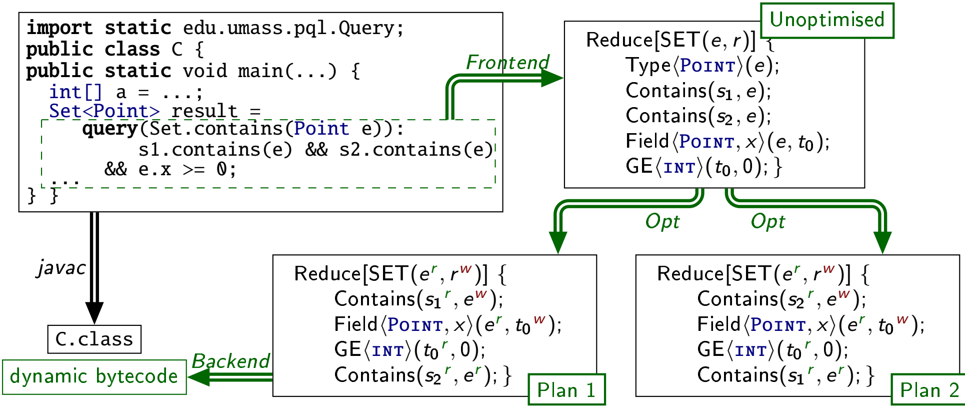
Dynamic Optimisation



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
  int[] a = ...;
  Set<Point> result =
     query(Set.contains(Point e)):
          s1.contains(e) && s2.contains(e)
       && e.x >= 0;
  ...
} }
```

Reduce[SET(e^r, r^w)] {
    Contains($s_1{}^r$, e^w);
    Field⟨POINT, x⟩(e^r, $t_0{}^w$);
    GE⟨INT⟩($t_0{}^r$, 0);
    Contains($s_2{}^r$, e^r); }

Dynamic Optimisation



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
  int[] a = ...;
  Set<Point> result =
     query(Set.contains(Point e)):
          s1.contains(e) && s2.contains(e)
       && e.x >= 0;
  ...
} }
```

Frontend

Unoptimised

$\mathrm{Reduce}[\mathrm{SET}(e, r)]\ \{$
$\quad \mathrm{Type}\langle \textsc{Point} \rangle(e);$
$\quad \mathrm{Contains}(s_1, e);$
$\quad \mathrm{Contains}(s_2, e);$
$\quad \mathrm{Field}\langle \textsc{Point}, x \rangle(e, t_0);$
$\quad \mathrm{GE}\langle \textsc{Int} \rangle(t_0, 0);\ \}$

Opt

Opt

$\mathrm{Reduce}[\mathrm{SET}(e^r, r^w)]\ \{$
$\quad \mathrm{Contains}({s_1}^r, e^w);$
$\quad \mathrm{Field}\langle \textsc{Point}, x \rangle(e^r, {t_0}^w);$
$\quad \mathrm{GE}\langle \textsc{Int} \rangle({t_0}^r, 0);$
$\quad \mathrm{Contains}({s_2}^r, e^r);\ \}$

Plan 1

$\mathrm{Reduce}[\mathrm{SET}(e^r, r^w)]\ \{$
$\quad \mathrm{Contains}({s_2}^r, e^w);$
$\quad \mathrm{Field}\langle \textsc{Point}, x \rangle(e^r, {t_0}^w);$
$\quad \mathrm{GE}\langle \textsc{Int} \rangle({t_0}^r, 0);$
$\quad \mathrm{Contains}({s_1}^r, e^r);\ \}$

Plan 2

javac

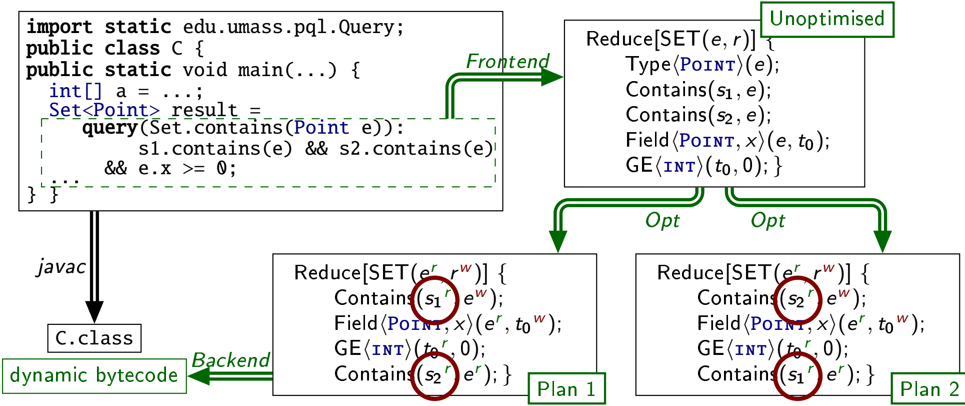
C.class

Backend

dynamic bytecode

- Compare different execution orders

Dynamic Optimisation



```
import static edu.umass.pql.Query;
public class C {
public static void main(...) {
  int[] a = ...;
  Set<Point> result =
     query(Set.contains(Point e)):
          s1.contains(e) && s2.contains(e)
       && e.x >= 0;
  ...
} }
```

Unoptimised:

$\text{Reduce}[\text{SET}(e, r)]\ \{$
$\quad \text{Type}\langle \textsc{Point}\rangle(e);$
$\quad \text{Contains}(s_1, e);$
$\quad \text{Contains}(s_2, e);$
$\quad \text{Field}\langle \textsc{Point}, x\rangle(e, t_0);$
$\quad \text{GE}\langle \textsc{Int}\rangle(t_0, 0);\ \}$

Plan 1:

$\text{Reduce}[\text{SET}(e^r, r^w)]\ \{$
$\quad \text{Contains}(s_1{}^r, e^w);$
$\quad \text{Field}\langle \textsc{Point}, x\rangle(e^r, t_0{}^w);$
$\quad \text{GE}\langle \textsc{Int}\rangle(t_0{}^r, 0);$
$\quad \text{Contains}(s_2{}^r, e^r);\ \}$

Plan 2:

$\text{Reduce}[\text{SET}(e^r, r^w)]\ \{$
$\quad \text{Contains}(s_2{}^r, e^w);$
$\quad \text{Field}\langle \textsc{Point}, x\rangle(e^r, t_0{}^w);$
$\quad \text{GE}\langle \textsc{Int}\rangle(t_0{}^r, 0);$
$\quad \text{Contains}(s_1{}^r, e^r);\ \}$

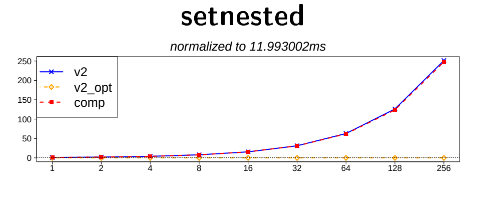
- Compare different execution orders
- Here: Iterate over s_1 or s_2?

Dynamic Optimisation: Extreme Cases

```
Set<Point> result =
    query(Set.contains(Point e)):
          s1.contains(e)
       && s2.contains(e)
       && e.x > 0;
```

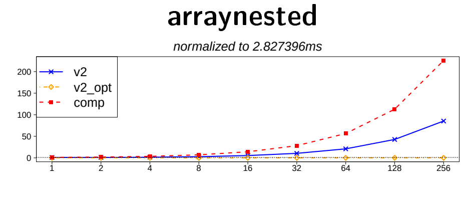
Dynamic Optimisation: Extreme Cases

```
Set<Point> result =
    query(Set.contains(Point e)):
          s1.contains(e)
       && s2.contains(e)
       && e.x > 0;
```



setnested

normalized to 11.993002ms

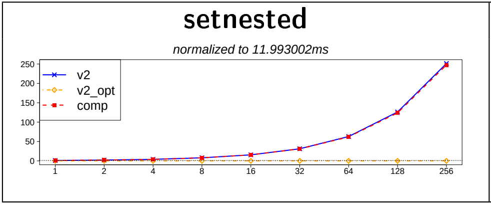


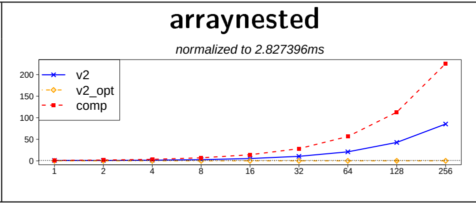
arraynested

normalized to 2.827396ms

Dynamic Optimisation: Extreme Cases

```
Set<Point> result =
    query(Set.contains(Point e)):
          s1.contains(e)
       && s2.contains(e)
       && e.x > 0;
```





Static compilation can't know whether s1 or s2 is bigger, can't optimise

Conclusions

- DSL for backend extensions:
  - Compact
  - Permits staged evaluation
- PQL backend rewrite:
  - Enabled dynamic optimisation
  - Dramatically faster whenever dynamic knowledge helps
  - Competitive when not
  - Dynamic recompilation time too high: $\Rightarrow$ JIT?
- Next steps:
  - New frontend (JastAdd)
  - Make PQL-ESL easier to use
    - Eliminate labels
    - Directly migrate Java bytecode to PQL-ESL
  - Easy-to-use rewriting formalism

Extension Example: SQL Access

DB(*spec*, *db*);
Table(*db*, `"tblname"`, *tbl*);
Column(*tbl*, `"column"`, *c*);
GT(*c*, 0);
Contains(*set*, *c*);

Extension Example: SQL Access

$\mathrm{DB}(\mathit{spec}, \mathit{db});$
$\mathrm{Table}(\mathit{db}, \texttt{"tblname"}, \mathit{tbl});$
$\mathrm{Column}(\mathit{tbl}, \texttt{"column"}, c);$
$\mathrm{GT}(c, 0);$
$\mathrm{Contains}(\mathit{set}, c);$

```
SELECT * FROM tblname;
```

transfer *ResultSet*

$$\text{filter: } \{c|\ \begin{aligned}[t] & v \in \mathit{ResultSet}, \\ & c = v.\texttt{column}, \\ & c > 0, c \in \mathit{set}\} \end{aligned}$$

Extension Example: SQL Access

DB(*spec*, *db*);
Table(*db*, `"tblname"`, *tbl*);
Column(*tbl*, `"column"`, *c*);
GT(*c*, 0);
Contains(*set*, *c*);

DB(*spec*, *db*);
TableQuery(*db*, *"tblname"*, *["column"]*,
[DBGT("column", 0)], tbl);
Column(*tbl*, *" column"* , *c*)
Contains(*set*, *c*)

```
SELECT * FROM tblname;
```

transfer *ResultSet*

$$\text{filter: } \{c \mid \begin{aligned} &v \in \mathit{ResultSet}, \\ &c = v.\texttt{column}, \\ &c > 0, c \in \mathit{set}\} \end{aligned}$$

Rewriting Support (WIP)

Extension Example: SQL Access

DB($spec, db$);
Table(db, "tblname", tbl);
Column(tbl, "column", c);
GT(c, 0);
Contains(set, c);

DB($spec, db$);
TableQuery(*db, "tblname", ["column"],*
[DBGT("column", 0)], tbl);
Column(tbl, " *column*" , c)
Contains(set, c)

```
SELECT * FROM tblname;
```

transfer *ResultSet*

filter: $\{c| \; v \in ResultSet, \; c = v.\texttt{column}, \; c > 0, c \in set\}$

```
SELECT column FROM tblname WHERE column > 0;
```

transfer *ResultSet*

filter: $\{c|c \in ResultSet, c \in set\}$

Rewriting Support (WIP)

Extension Example: SQL Access

DB($spec$, db);
Table(db, "tblname", tbl);
Column(tbl, "column", c);
GT(c, 0);
Contains(set, c);

DB($spec$, db);
TableQuery(*db, "tblname", ["column"],*
[DBGT("column", 0)], tbl);
Column(tbl," *column*" , c)
Contains(set, c)

```
SELECT * FROM tblname;
```

transfer *ResultSet*

filter: $\{c|\; v \in ResultSet,$
$c = v.\texttt{column},$
$c > 0, c \in set\}$

```
SELECT column FROM tblname WHERE column > 0;
```

transfer *ResultSet*

filter: $\{c|c \in ResultSet, c \in set\}$

Rewriting Support (WIP)

Query Planning Strategy

- Dynamic Programming selects shortest path
- Cost model:
  - *cost*: how much does one option cost?
  - *size*: how many options will we iterate over?
  - *selectivity*: how likely will past bindings match?
  - *parallel*: is this relation parallelisable?

    ⇒ discounts *size* when applicable

Query Planning Strategy

- Dynamic Programming selects shortest path
- Cost model:
  - *cost*: how much does one option cost?
  - *size*: how many options will we iterate over?
  - *selectivity*: how likely will past bindings match?
  - *parallel*: is this relation parallelisable?
    $\Rightarrow$ discounts *size* when applicable

- c: aggregate cost
- s: aggregate size and selectivity

$$\begin{aligned} c' &= c + \text{cost} \times s \\ s' &= s' \times \text{size} \times \text{selectivity} \end{aligned}$$

Predicting execution cost

```
cost_formula:
    formula = "1.0";
    __cost_formula = formula;

proceed_formula: // = size × selectivity
    if (isMode( (obj, key), (rr) )) {
        if ( isMode( (value), (r) ) )
             formula = "0.004";
        else formula = "1.0";
    } else {
        if (isConst(obj)) {
            if (isMode( (obj, key, value), (rwr || r_r) ))
                 formula = "obj[size]/250";
            else formula = "obj[size]";
        } else {
            if (isMode( (obj, key, value), (rwr || r_r) ))
                 formula = "100/250";
            else formula = "100";
    }    }
    __proceed_formula = formula;
```