A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel

IBM Research

ECOOP ’15

JRules example

```
rule FindMarketers {
    when {
        C: Client();
        Ms: aggregate {
            M: Marketer(clients.contains(C.id));
        }  do { collect {M}; }
    } then {
        insert new C2Ms(C, Ms);
    }
}
```

```
class Marketer {
    List<Client> clients;
}
class Client {
    int id;
}
```

```
class C2MS {
    Client C;
    List<Marketer> Ms;
}
```

Calculus for **A**ggregating **M**atching **P**atterns

$p ::= d$	*constant data*
$\mid \oplus\ p$	*unary operator*
$\mid p_1 \otimes p_2$	*binary operator*
$\mid$ **map** p	*map over a bag*
$\mid$ **assert** p	*assertion*
$\mid p_1 \mid\mid p_2$	*error recovery (orElse)*
$\mid$ **it** $\mid$ **let it** $= p_1$ **in** p_2	*get/set scrutinee*
$\mid$ **env** $\mid$ **let env** $+= p_1$ **in** p_2	*get/update environment*

Calculus for **A**ggregating **M**atching **P**atterns

$$\begin{aligned} & \text{it.}\textit{type} = \text{“Marketer”} \\ \wedge\ & \text{env.C.}\textit{data.id} \in \text{it.}\textit{data.clients} \\ \wedge\ & \text{let env += [M : it] in env} \end{aligned}$$

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    }  do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Rules

r ::= **when** p; r	*Evaluate p against each WME*
| **global** p; r	*Evaluate p once*
| **not** p; r	*Ensure p does hold for any WME*
| **return** p	*Compute a result using p*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| $\oplus$ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\boxed{\sigma \vdash p \;@\; d \Downarrow_r d?}$

Unary Operators

$\oplus$ d ::=

| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\boxed{\oplus\, d \Downarrow_o d}$

Binary Operators

$d_1 \otimes d_2$ ::=

| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$\boxed{d \otimes d \Downarrow_o d}$

JRules Engine

CAMP semantics

✔

Nested **R**elational **A**lgebra

$q ::= d$	*constant data*
$\mid$ **In**	*context value*
$\mid \oplus\, q$	*unary operator*
$\mid q_1 \otimes q_2$	*binary operator*
$\mid \chi\langle q_2\rangle(q_1)$	*map*
$\mid \sigma\langle q_2\rangle(q_1)$	*select*
$\mid q_1 \times q_2$	*cartesian product*
$\mid \bowtie^{d}\langle q2\rangle(q_1)$	*dependent join*
$\mid q_1 \mid\mid q_2$	*default*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| $\oplus$ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\boxed{\sigma \vdash p \; @ \; d \Downarrow_r d?}$

Unary Operators

$\oplus$ d ::=

| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\boxed{\oplus\, d \Downarrow_o d}$

Binary Operators

$d_1 \otimes d_2$ ::=

| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$\boxed{d \otimes d \Downarrow_o d}$

JRules Engine

CAMP semantics

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| $\oplus$ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

$\boxed{q \; @ \; d \Downarrow_a d}$

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
$\mid \oplus$ p	*unary operator*
$\mid p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
$\mid p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \,@\, d \Downarrow_r d?$

Unary Operators

$\oplus$ d ::=

| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

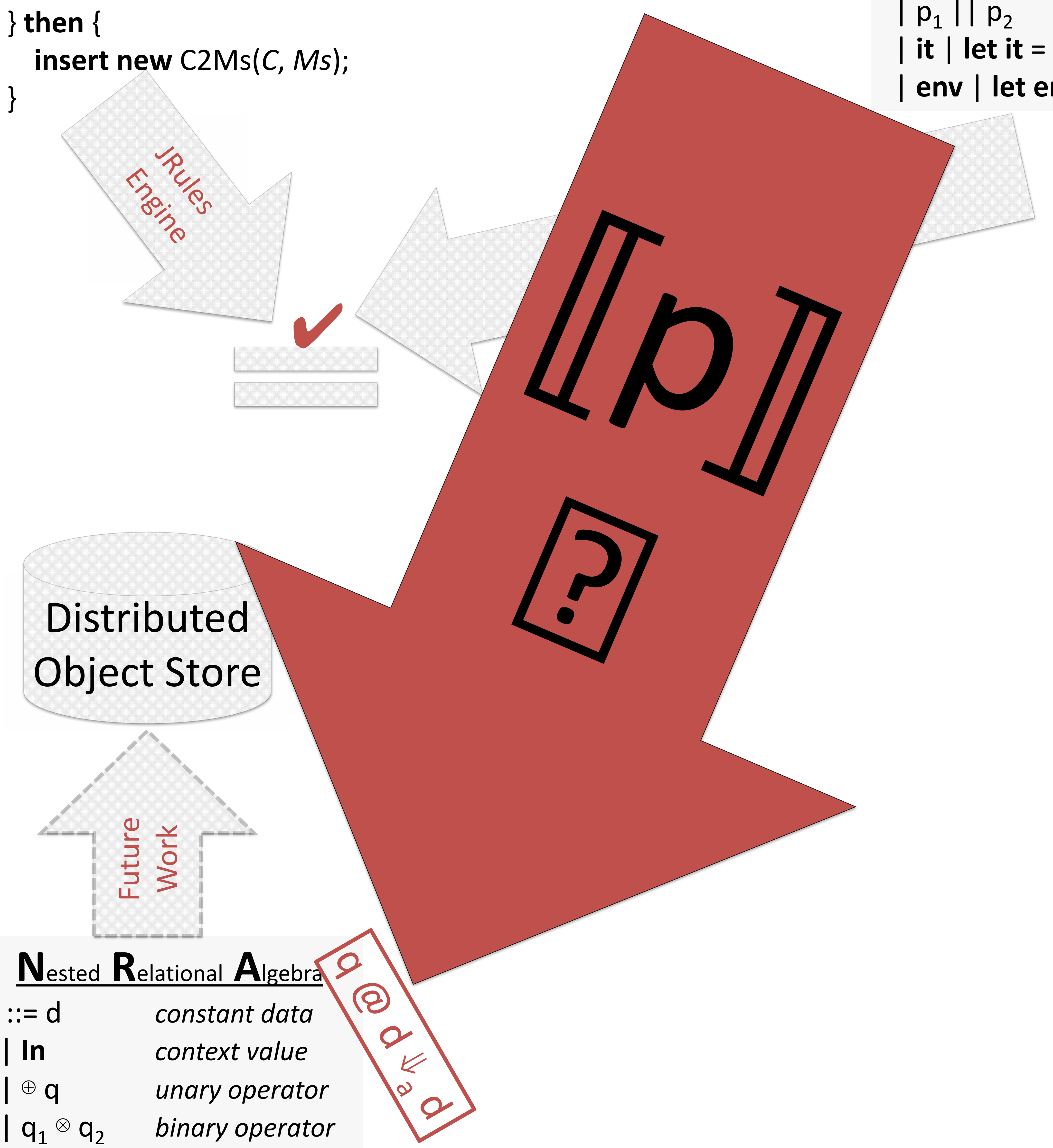
$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=

$\mid d_1 = d_2$	*equality*
$\mid d_1 \in d_2$	*element of*
$\mid d_1 \cup d_2$	*union*
$\mid d_1 * d_2$	*biased record concat*
$\mid d_1 + d_2$	*compatible record concat*

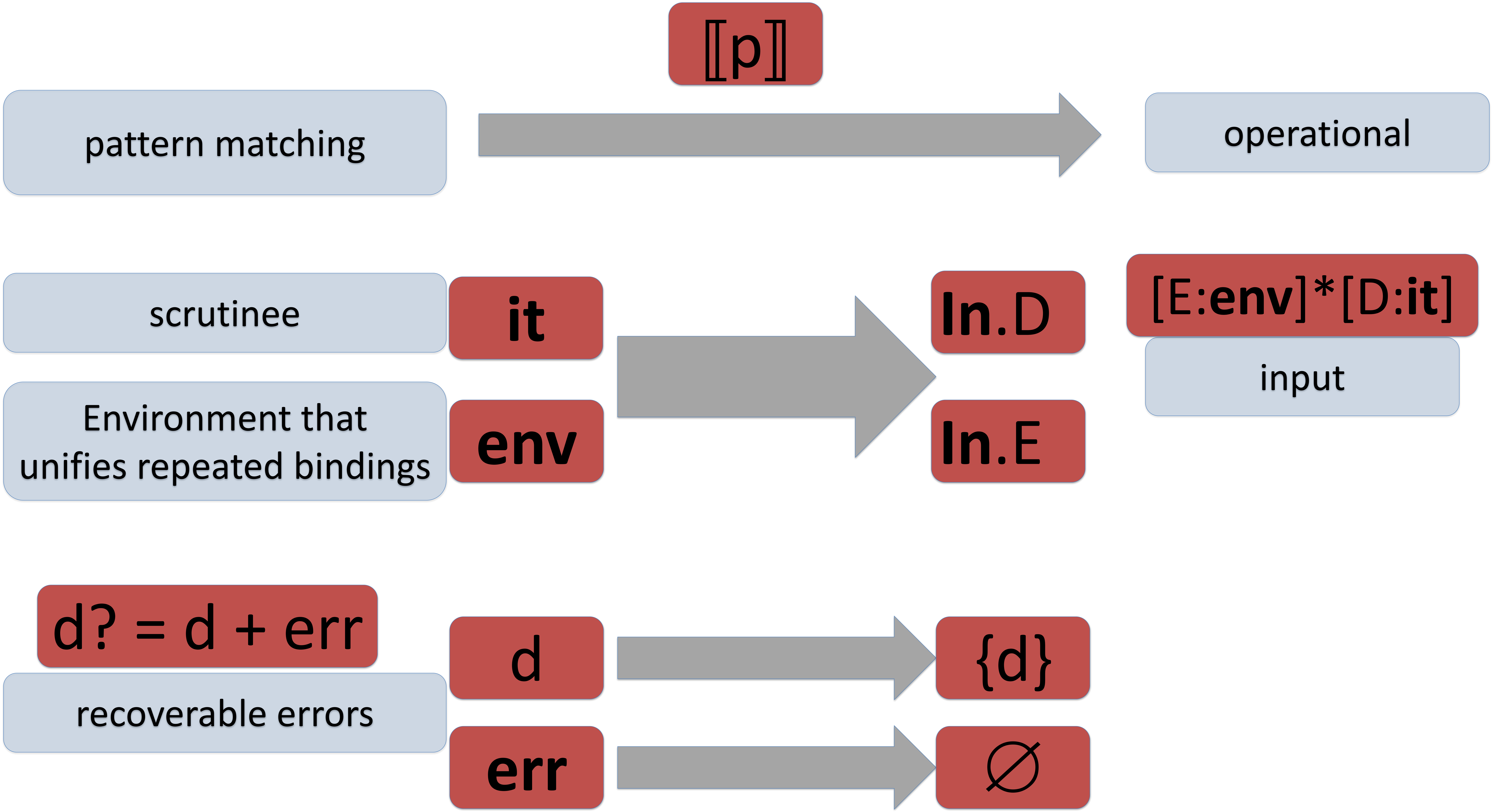
$d \otimes d \Downarrow_o d$



Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
$\mid \oplus$ q	*unary operator*
$\mid q_1 \otimes q_2$	*binary operator*
$\mid \chi\langle q_2\rangle(q_1)$	*map*
$\mid \sigma\langle q_2\rangle(q_1)$	*select*
$\mid q_1 \times q_2$	*cartesian product*
$\mid \bowtie^d\langle q2\rangle(q_1)$	*dependent join*
$\mid q_1 \mid\mid q_2$	*default*

CAMP → NRA



[[p]]
pattern matching
operational
scrutinee
it
Environment that unifies repeated bindings
env
In.D
In.E
[E:env]*[D:it]
input
d? = d + err
recoverable errors
d
{d}
err
∅

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| p_1 || p_2	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p\ @\ d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\ d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

JRules Engine

CAMP semantics

⟦p⟧□ @ [E:σ]*[D:d_1] $\Downarrow_a$ {d_2} ↔ σ ⊢ p @ $d_1 \Downarrow_r d_2$
⟦p⟧□ @ [E:σ]*[D:d_1] $\Downarrow_a$ ∅ ↔ σ ⊢ p @ $d_1 \Downarrow_r$ err

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $⋈^d\langle q2\rangle(q_1)$	*dependent join*
| q_1 || q_2	*default*

q @ d

Compilation is Semantics Preserving

$$[\![p]\!]□\ @\ [E:\sigma]*[D:d_1] \Downarrow_a \{d_2\} \leftrightarrow \sigma \vdash p\ @\ d_1 \Downarrow_r d_2$$

$$[\![p]\!]□\ @\ [E:\sigma]*[D:d_1] \Downarrow_a \varnothing \leftrightarrow \sigma \vdash p\ @\ d_1 \Downarrow_r \textbf{err}$$

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Artifact Evaluated · ECOOP · AEC · Consistent · Complete · Well Documented · Easy to Reuse

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| $\oplus$ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\boxed{\sigma \vdash p \mathbin{@} d \Downarrow_r d?}$

Unary Operators

$\oplus$ d ::=

| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\boxed{\oplus\, d \Downarrow_o d}$

Binary Operators

$d_1 \otimes d_2$ ::=

| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$\boxed{d \otimes d \Downarrow_o d}$

JRules Engine

CAMP semantics

$[\![p]\!]^{?} \mathbin{@} [E{:}\sigma]*[D{:}d_1] \Downarrow_a \{d_2\} \leftrightarrow \sigma \vdash p \mathbin{@} d_1 \Downarrow_r d_2$

$[\![p]\!]^{?} \mathbin{@} [E{:}\sigma]*[D{:}d_1] \Downarrow_a \emptyset \leftrightarrow \sigma \vdash p \mathbin{@} d_1 \Downarrow_r \mathbf{err}$

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| $\oplus$ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

$\boxed{q \mathbin{@} d \Downarrow_a d}$

Named Nested Relational Calculus

$e ::= x$	*variables*
$\mid d$	*constant data*
$\mid \oplus\ e$	*unary operator*
$\mid e_1 \otimes e_2$	*binary operator*
$\mid \mathbf{let}\ x{=}e_1\ \mathbf{in}\ e_2$	*let*
$\mid \{e_2 \mid x \in e_1\}$	*comprehension*
$\mid e_1\ ?\ e_2 : e_3$	*conditional*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

JRules Engine

CAMP semantics

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| p_1 || p_2	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \;@\; d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

$[\![p]\!] @ [E:\sigma]*[D:d_1] \Downarrow_a \{d_2\} \leftrightarrow \sigma \vdash p \;@\; d_1 \Downarrow_r d_2$

$[\![p]\!] @ [E:\sigma]*[D:d_1] \Downarrow_a \emptyset \leftrightarrow \sigma \vdash p \;@\; d_1 \Downarrow_r \mathbf{err}$

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| q_1 || q_2	*default*

$q \;@\; d \Downarrow_a d$

$[\![q]\!]_x$

$\sigma \vdash e \Downarrow_c d$

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| e_1 ? e_2 : e_3	*conditional*

NRA → NNRC

$[\![q]\!]_x$

operational → declarative

input **In** → **x** environment

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

JRules Engine

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \;@\; d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

Compilation is Semantics Preserving

$$\text{if } \sigma(x) = d_1$$

$$q \;@\; d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [\![q]\!]_x ? \Downarrow_c d_2$$

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

$q \;@\; d \Downarrow_a d$

if $\sigma(x) = d_1$
$q \;@\; d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [\![q]\!]_x ? \Downarrow_c d_2$

$\sigma \vdash e \Downarrow_c d$

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| $e_1\ ?\ e_2 : e_3$	*conditional*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

JRules Engine

CAMP semantics

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \,@\, d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

$[\![p]\!]^{?} \,@\, [E{:}\sigma]*[D{:}d_1] \Downarrow_a \{d_2\} \leftrightarrow \sigma \vdash p \,@\, d_1 \Downarrow_r d_2$

$[\![p]\!]^{?} \,@\, [E{:}\sigma]*[D{:}d_1] \Downarrow_a \emptyset \leftrightarrow \sigma \vdash p \,@\, d_1 \Downarrow_r err$

$[\![e]\!]^{?}$

Distributed Object Store

Future Work

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

$q \,@\, d \Downarrow_a d$

if $\sigma(x) = d_1$
$q \,@\, d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [\![q]\!]_x^{?} \Downarrow_c d_2$

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| $e_1\ ?\ e_2 : e_3$	*conditional*

NNRC → CAMP

⟦e⟧
declarative
pattern matching
scrutinee
renaming
environment
x
env.x
Environment that unifies repeated bindings
recoverable errors

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel

IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Compiler

JRules Engine

CAMP semantics

Distributed Object Store

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \;@\; d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

$\sigma \vdash p \;@\; d_1 \Downarrow_r d_2 \leftrightarrow \sigma \vdash p \;@\; [E{:}\sigma]*[D{:}d_1] \Downarrow_a \{d_2\}$

$\sigma \vdash p \;@\; d_1 \Downarrow_r err \leftrightarrow \sigma \vdash p \;@\; [E{:}\sigma]*[D{:}d_1] \Downarrow_a \emptyset$

$\sigma \vdash [\![e]\!] \;@\; d_0 \Downarrow_r d \leftrightarrow \sigma \vdash e \Downarrow_c d$

$\sigma \vdash e \Downarrow_c d$

Compilation is Semantics Preserving

$$\sigma \vdash [\![e]\!] \;@\; d_0 \Downarrow_r d \leftrightarrow \sigma \vdash e \Downarrow_c d$$

$\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
$q_1 \mid\mid q_2$	*default*

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** $x=e_1$ **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| $e_1 \;?\; e_2 : e_3$	*conditional*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

rule FindMarketers {
  when {
    C: Client();
    Ms: **aggregate** {
      M: Marketer(clients.contains(*C*.id));
    } **do** { collect {*M*}; }
  } **then** {
    insert new C2Ms(*C*, *Ms*);
  }
}

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p @ d \Downarrow_r d?$

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

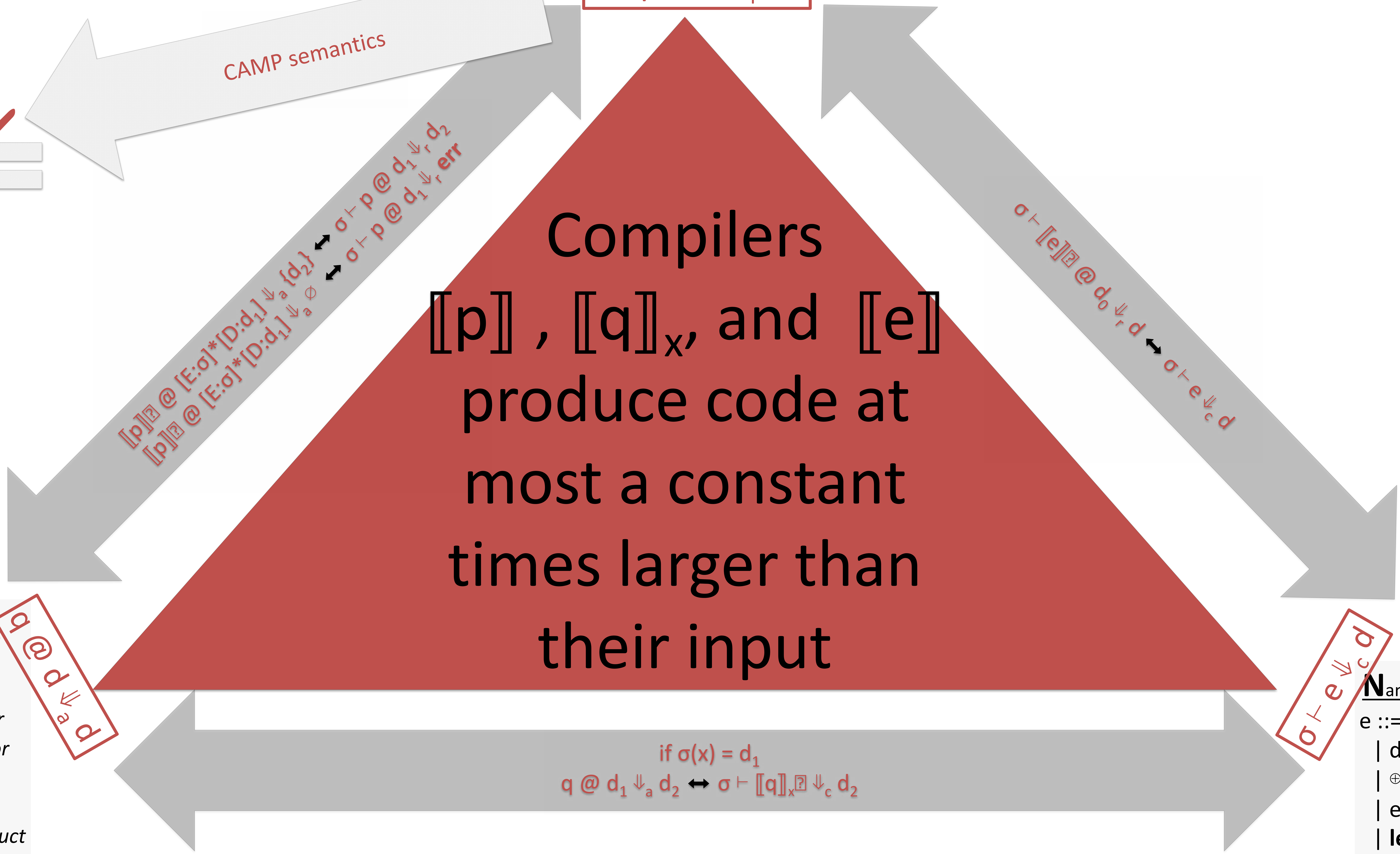
$d \otimes d \Downarrow_o d$

JRules Engine

CAMP semantics

Distributed Object Store

Future Work



$q @ d \Downarrow_a d$

$\sigma \vdash e \Downarrow_c d$

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $⋈^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| {e_2 | x ∈ e_1}	*comprehension*
| e_1 ? e_2 : e_3	*conditional*

A Pattern Calculus for Rule Languages:
Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research
Everything in gray has been verified in Coq
Distinguished Artifact Award
IBM JRules
rule FindMarketers {
when {
C: Client();
Ms: aggregate {
M: Marketer(clients.contai
} do { collect {M}; }
} then {
insert new C2Ms(C, Ms);
}
}
JRules Engine
Rules
Calculus for Aggregating Matching Patterns
constant data
operator
operator
er a bag
on
ecovery (orElse)
scrutinee
date environment
Unary Operators
⊕ d ::=
| identity d — no-op. returns d
| ¬d — negates a Boolean
| {d} — singleton bag of d
| #d — size of bag
| flatten d — flatten a bag of bags
| [A:d] — record constructor
| d.A — field selection
| d−A — field removal
⊕ d ⇓o d
Binary Operators
d1 ⊗ d2 ::=
| d1 = d2 — equality
| d1 ∈ d2 — element of
| d1 ∪ d2 — union
| d1 * d2 — biased record concat
| d1 + d2 — compatible record concat
d ⊗ d ⇓o d
Γ ⊢ p :r τ0 → τ1
⊕:o τ0 → τ1
⊗:o τ0 → τ1 → τ2
Distributed Object Store
Future Work
Nested Relational Algebra
q ::= d — constant data
| In — context value
| ⊕ q — unary operator
| q1 ⊗ q2 — binary operator
| χ⟨q2⟩(q1) — map
| σ⟨q2⟩(q1) — select
| q1 ×q2 — cartesian product
| ⋈d⟨q2⟩(q1) — dependent join
| q1 || q2 — default
q @ d ⇓a d
Named Nested Relational Calculus
e ::= x — variables
| d — constant data
| ⊕ e — unary operator
| e1 ⊗ e2 — binary operator
| let x=e1 in e2 let
| {e2 | x ∈ e1} comprehension
| e1 ? e2 : e3 — conditional
σ ⊢ e
if σ(x) = d1
q @ d1 ⇓a d2 ↔ σ ⊢ [[q]]x ⇓c d2
ECOOP 2015
Slide 20

A Pattern Calculus for Rule Languages:

…, Compilation, and Mechanization

…n Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

Type Soundness

$\sigma :_d \Gamma$

$d_0 :_d \tau_0$

$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$

implies that $\exists\, d_1?,$

$\sigma \vdash p \;@\; d_0 \Downarrow_r d_1?$

$d_1? :_d \tau_1$

rule Fi…
whe…
C:
Ms…
} c…
} the…
ins…
}
}

…lculus for **A**ggregating **M**atching **P**atterns

	constant data
	unary operator
	binary operator
	map over a bag
…p	*assertion*
…$_2$	*error recovery (orElse)*
…it = p_1 **in** p_2	*get/set scrutinee*
…et env += p_1 **in** p_2	*get/update environment*

$\sigma \vdash p \;@\; d \Downarrow_r d?$

$\sigma :_d \Gamma$
$d_0 :_d \tau_0$
$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$
implies that $\exists\, d_1?,$
$\sigma \vdash p \;@\; d_0 \Downarrow_r d_1?$
$d_1? :_d \tau_1$

$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$

Unary Operators

$\oplus\, d$ ::=

| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

$\oplus\, d \Downarrow_o d$

$d_0 :_d \tau_0$
$\oplus :_o \tau_0 \rightarrow \tau_1$
implies that $\exists\, d_1,$
$\oplus\, d_0 \Downarrow_o d_1$
$d_1 :_d \tau_1$

$\oplus :_o \tau_0 \rightarrow \tau_1$

Binary Operators

$d_1 \otimes d_2$ ::=

| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

$d_0 :_d \tau_0$
$d_1 :_d \tau_1$
$\otimes :_o \tau_0 \rightarrow \tau_1 \rightarrow \tau_2$
implies that $\exists\, d_2,$
$d_0 \otimes d_1 \Downarrow_o d_2$
$d_2 :_d \tau_2$

$\otimes :_o \tau_0 \rightarrow \tau_1 \rightarrow \tau_2$

$\sigma \vdash [\![e]\!] @ d_0 \Downarrow_r d \leftrightarrow \sigma \vdash e \Downarrow_c d$

D…
O…

Futu… Wo…

Compilers
$[\![p]\!]$, $[\![q]\!]_x$, and $[\![e]\!]$
produce code at most a constant times larger than their input

Nested **R**elational **A**lgebra

q ::= d	*constant data*
| **In**	*context value*
| $\oplus$ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \;\|\|\; q_2$	*default*

$q \;@\; d \Downarrow_a d$

if $\sigma(x) = d_1$
$q \;@\; d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [\![q]\!]_x \Downarrow_c d_2$

$\sigma \vdash e \Downarrow_c d$

Named **N**ested **R**elational **C**alculus

e ::= x	*variables*
| d	*constant data*
| $\oplus$ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| e_1 ? e_2 : e_3	*conditional*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
```

Compiler

Rules

r ::= **when** p; r
| **global** p; r
| **not** p; r
| **return** p

Calculus for Aggregating Matching Patterns

p ::= d	*constant data*
| ⊕ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| p_1 || p_2	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

Unary Operators

⊕ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

$d \otimes d \Downarrow_o d$

$\oplus d \Downarrow_o d$

$\sigma \vdash p @ d \Downarrow_r d?$

$\sigma :_d \Gamma$
$d_0 :_d \tau_0$
$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$
implies that $\exists d_1?$,
$\sigma \vdash p @ d_0 \Downarrow_r d_1?$
$d_1? :_d \tau_1$

$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$

$$q :_a \tau_0 \rightarrow \tau_1$$

$$\Gamma \vdash e :_c \tau$$

Distributed Object Store

Future Work

Compilers ⟦p⟧, ⟦q⟧$_x$, and ⟦e⟧ produce code at most a constant times larger than their input

$q :_a \tau_0 \rightarrow \tau_1$

$q @ d \Downarrow_a d$

$\Gamma \vdash e :_c \tau$

$\sigma \vdash e \Downarrow_c d$

Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| q_1 || q_2	*default*

if $\sigma(x) = d_1$
$q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [\![q]\!]_x \Downarrow_c d_2$

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| {e_2 | x ∈ e_1}	*comprehension*
| e_1 ? e_2 : e_3	*conditional*

Type Soundness
$d_0 :_d \tau_0$
$q :_a \tau_0 \rightarrow \tau_1$
implies that $\exists d_1,$
$q \ @ \ d_0 \Downarrow_a d_1$
$d_1 :_d \tau_1$
Type Soundness
$\sigma :_d \Gamma$
$\Gamma \vdash e :_c \tau$
implies that $\exists d,$
$\sigma \vdash e \Downarrow_c d$
$d :_d \tau$
IBM Research
constant data
unary operator
binary operator
map over a bag
assertion
$\sigma \vdash p \ @ \ d \Downarrow_r d?$
$\sigma :_d \Gamma$
$d_0 :_d \tau_0$
$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$
implies that $\exists d_1?,$
$\sigma \vdash p \ @ \ d_0 \Downarrow_r d_1?$
$d_1? :_d \tau_1$
$\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$
Distributed Object Store
Future Work
Compilers
⟦p⟧, ⟦q⟧$_x$, and ⟦e⟧
produce code at
most a constant
times larger than
their input
Nested Relational Algebra
q ::= d — constant data
| In — context value
| ⊕ q — unary operator
| $q_1 \otimes q_2$ — binary operator
| $\chi\langle q_2\rangle(q_1)$ — map
| $\sigma\langle q_2\rangle(q_1)$ — select
| $q_1 \times q_2$ — cartesian product
| $\bowtie^d\langle q2\rangle(q_1)$ — dependent join
| $q_1 \ || \ q_2$ — default
if $\sigma(x) = d_1$
$q \ @ \ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash ⟦q⟧_x \Downarrow_c d_2$
Named Nested Relational Calculus
e ::= x — variables
| d — constant data
| ⊕ e — unary operator
| $e_1 \otimes e_2$ — binary operator
| **let** $x = e_1$ **in** e_2 — let
| $\{e_2 \mid x \in e_1\}$ — comprehension
| $e_1 \ ? \ e_2 : e_3$ — conditional
ECOOP 2015

A Pattern Calculus for Rule Languages:

Everything in

Distinguished Artifact Award

Type Preservation

$$[\![p]\!] :_a [E:\Gamma, D:\tau_0] \to \{\tau_1\} \leftrightarrow \Gamma \vdash p :_r \tau_0 \to \tau_1$$

Nested **R**elational **A**lgebra

q ::= d	*constant data*
| **In**	*context value*
| ⊕ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $⋈^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

Named **N**ested **R**elational **C**alculus

e ::= x	*variables*
| d	*constant data*
| ⊕ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** $x=e_1$ **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| $e_1\ ?\ e_2 : e_3$	*conditional*

A Pattern Calculus for Rule Languages:

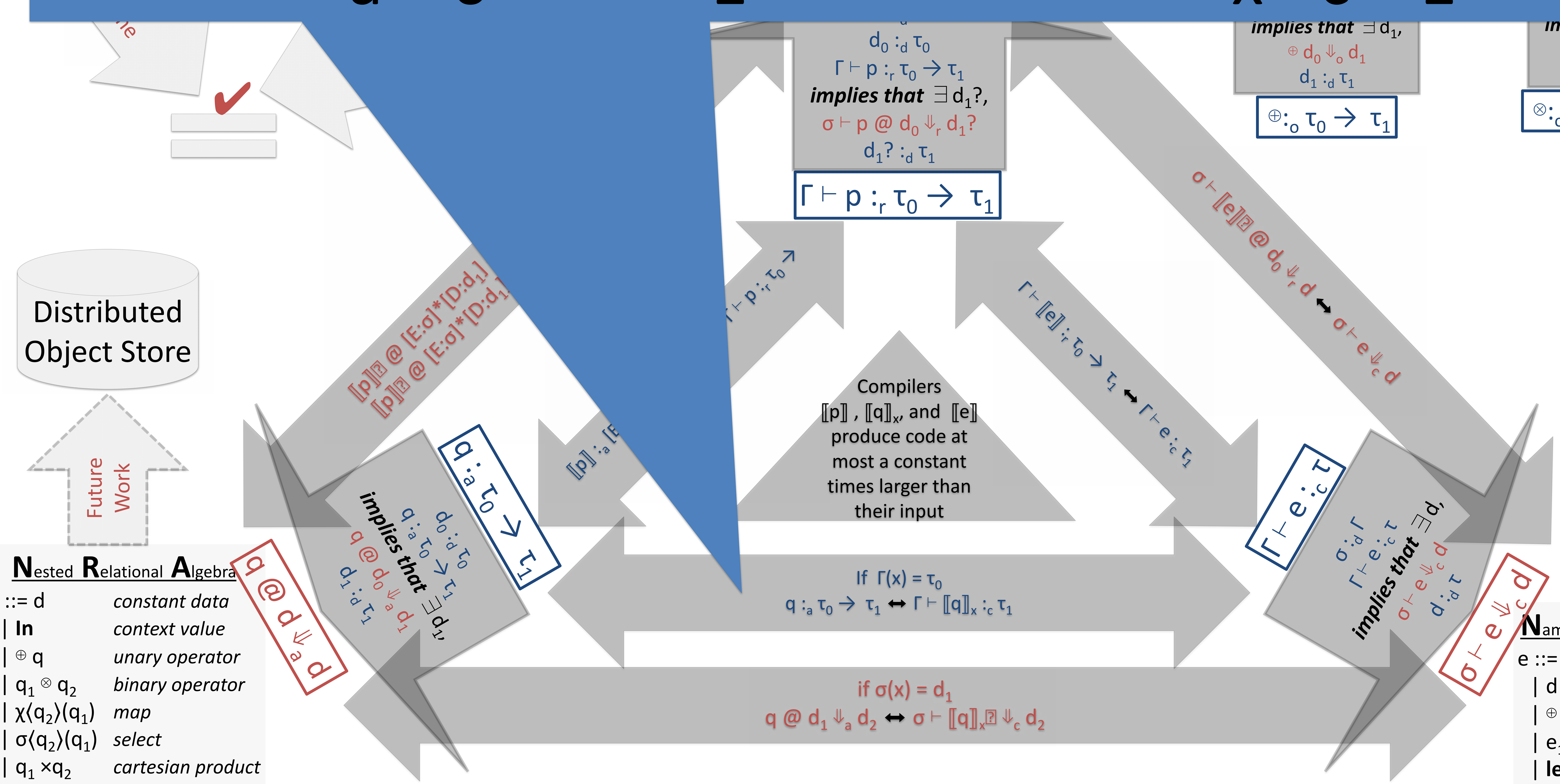
Everything in

Distinguished Artifact Award

Type Preservation

$$\text{If } \Gamma(x) = \tau_0$$

$$q :_a \tau_0 \to \tau_1 \leftrightarrow \Gamma \vdash [\![q]\!]_x :_c \tau_1$$



Nested **R**elational **A**lgebra

q ::= d	*constant data*
| **In**	*context value*
| $\oplus$ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

Named **N**ested **R**elational **C**alculus

e ::= x	*variables*
| d	*constant data*
| $\oplus$ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| e_1 ? e_2 : e_3	*conditional*

A Pattern Calculus for Rule Languages:

Polymorphic type inference for NNRC
- is NP-complete [1]
- has a constraint based algorithm [1]

Since type preservation is bi-directional, this result and algorithm applies to CAMP (and NRA) as well.

[1] J. Van den Bussche and S. Vansummeren. Polymorphic type inference for the named nested rela-tional calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.

Everything in gray has been verified in Coq

Distinguished Artifact Award

Binary Operators

$d_1 \otimes d_2$ ::=
| $d_1 = d_2$ *equality*
| $d_1 \in d_2$ *element of*
| $d_1 \cup d_2$ *union*
| $d_1 * d_2$ *biased record concat*
| $d_1 + d_2$ *compatible record concat*

$\boxed{d \otimes d \Downarrow_o d}$

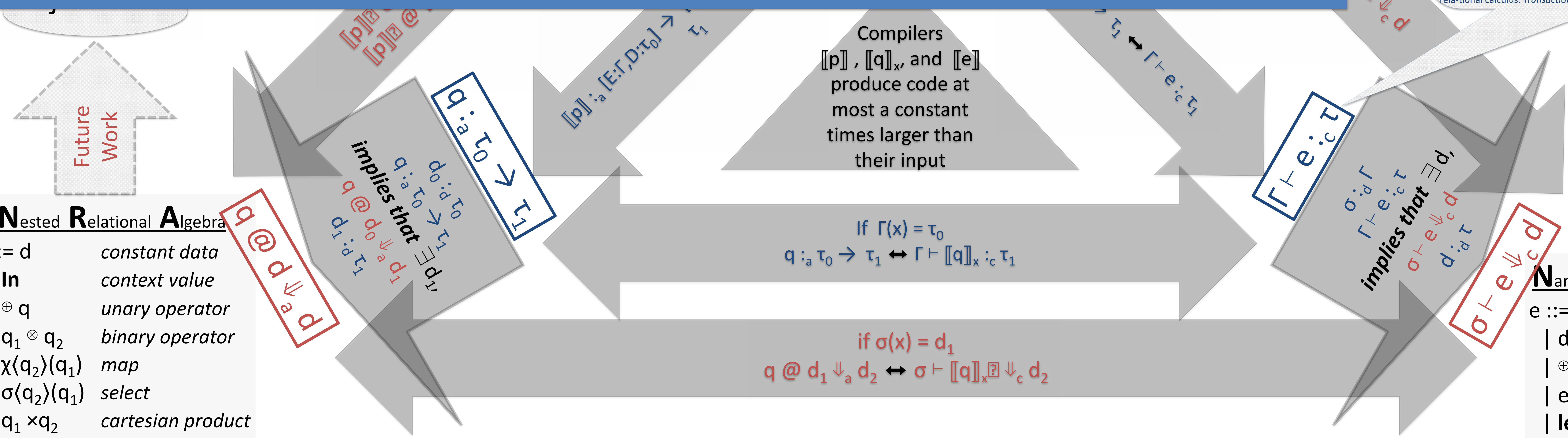
$d_0 :_d \tau_0$
$d_1 :_d \tau_1$
$\otimes :_o \tau_0 \to \tau_1 \to \tau_2$
implies that $\exists d_2$,
$d_0 \otimes d_1 \Downarrow_o d_2$
$d_2 :_d \tau_2$

$\boxed{\otimes :_o \tau_0 \to \tau_1 \to \tau_2}$

Polymorphic type inference for NNRC
- is NP-complete [1]
- has a constraint based algorithm [1]

Since type preservation is bi-directional, this result and algorithm applies to CAMP (and NRA) as well.

[1] J. Van den Bussche and S. Vansummeren. Polymorphic type inference for the named nested rela-tional calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.



Nested Relational Algebra

q ::= d *constant data*
| **In** *context value*
| $\oplus$ q *unary operator*
| $q_1 \otimes q_2$ *binary operator*
| $\chi\langle q_2\rangle(q_1)$ *map*
| $\sigma\langle q_2\rangle(q_1)$ *select*
| $q_1 \times q_2$ *cartesian product*
| $\bowtie^d\langle q2\rangle(q_1)$ *dependent join*
| $q_1 \mid\mid q_2$ *default*

Named Nested Relational Calculus

e ::= x *variables*
| d *constant data*
| $\oplus$ e *unary operator*
| $e_1 \otimes e_2$ *binary operator*
| **let** x=e_1 **in** e_2 *let*
| $\{e_2 \mid x \in e_1\}$ *comprehension*
| e_1 ? e_2 : e_3 *conditional*

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization

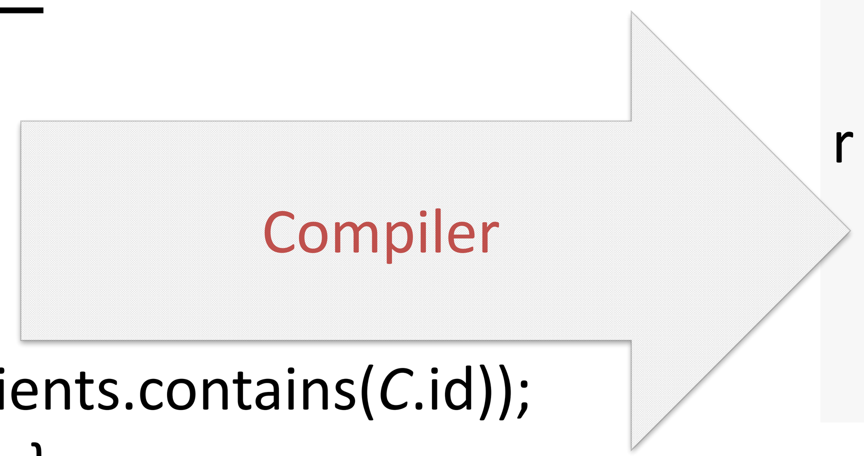
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Everything in gray has been verified in Coq

Distinguished Artifact Award

IBM JRules

```
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```



Rules

r ::= **when** p; r	
| **global** p; r	
| **not** p; r	
| **return** p	

Calculus for Aggregating Matching Patterns

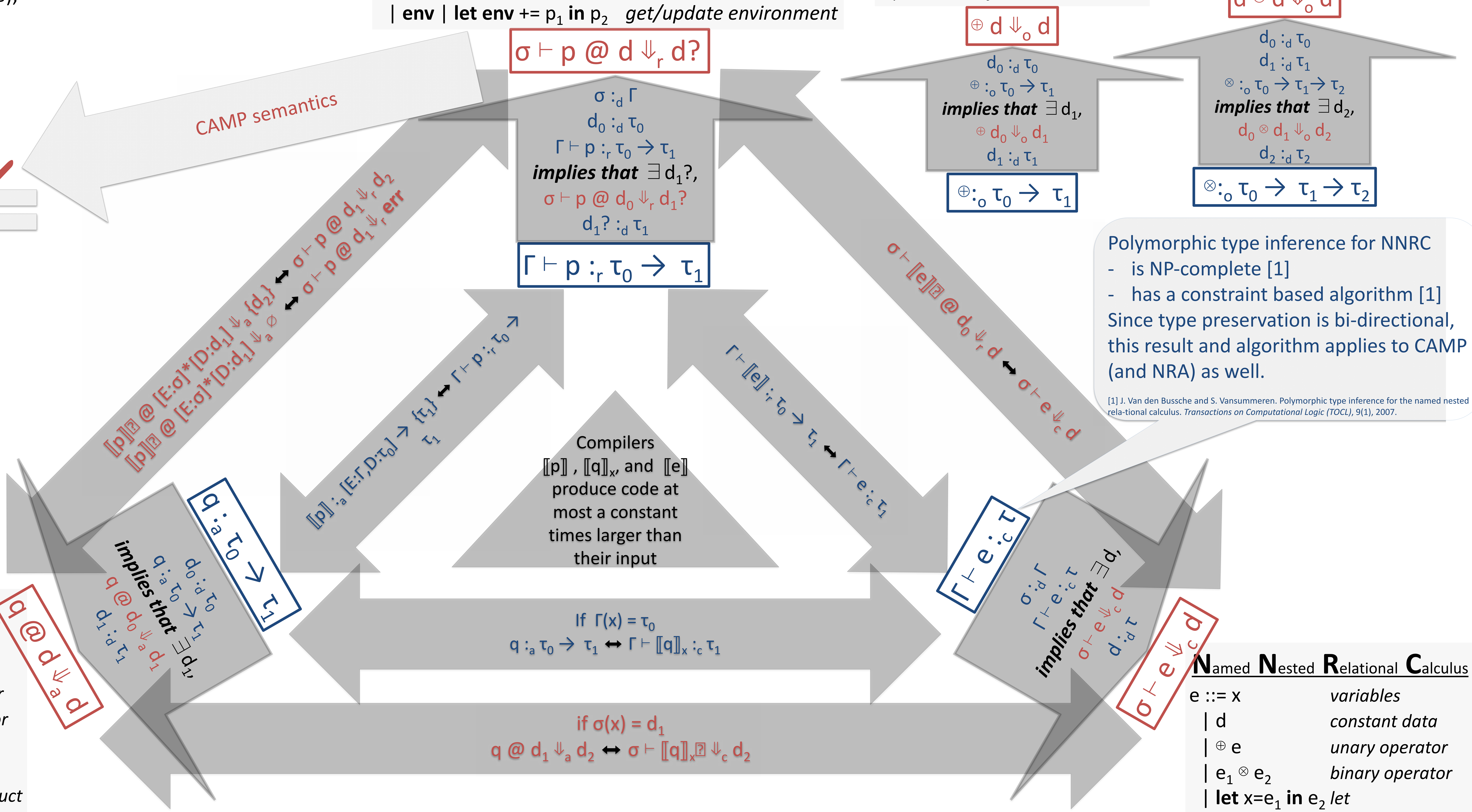
p ::= d	*constant data*
| $\oplus$ p	*unary operator*
| $p_1 \otimes p_2$	*binary operator*
| **map** p	*map over a bag*
| **assert** p	*assertion*
| $p_1 \mid\mid p_2$	*error recovery (orElse)*
| **it** | **let it** = p_1 **in** p_2	*get/set scrutinee*
| **env** | **let env** += p_1 **in** p_2	*get/update environment*

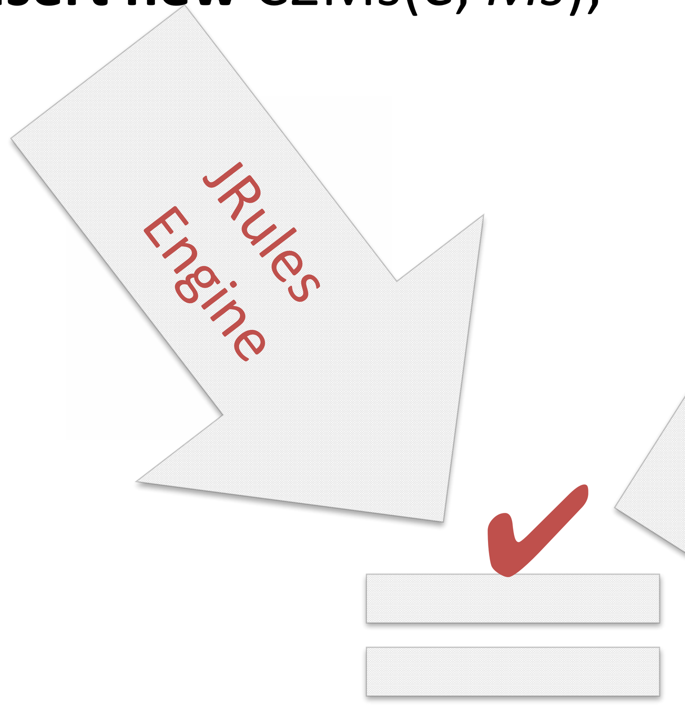
Unary Operators

$\oplus$ d ::=	
| **identity** d	*no-op. returns d*
| ¬d	*negates a Boolean*
| {d}	*singleton bag of d*
| #d	*size of bag*
| **flatten** d	*flatten a bag of bags*
| [A:d]	*record constructor*
| d.A	*field selection*
| d−A	*field removal*

Binary Operators

$d_1 \otimes d_2$ ::=	
| $d_1 = d_2$	*equality*
| $d_1 \in d_2$	*element of*
| $d_1 \cup d_2$	*union*
| $d_1 * d_2$	*biased record concat*
| $d_1 + d_2$	*compatible record concat*

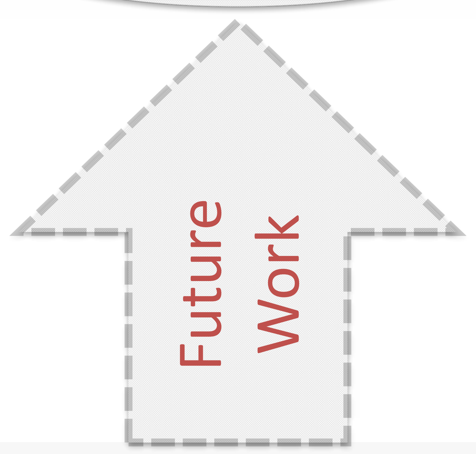




Polymorphic type inference for NNRC
- is NP-complete [1]
- has a constraint based algorithm [1]

Since type preservation is bi-directional, this result and algorithm applies to CAMP (and NRA) as well.

[1] J. Van den Bussche and S. Vansummeren. Polymorphic type inference for the named nested rela-tional calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.



Nested Relational Algebra

q ::= d	*constant data*
| **In**	*context value*
| $\oplus$ q	*unary operator*
| $q_1 \otimes q_2$	*binary operator*
| $\chi\langle q_2\rangle(q_1)$	*map*
| $\sigma\langle q_2\rangle(q_1)$	*select*
| $q_1 \times q_2$	*cartesian product*
| $\Join^d\langle q2\rangle(q_1)$	*dependent join*
| $q_1 \mid\mid q_2$	*default*

Named Nested Relational Calculus

e ::= x	*variables*
| d	*constant data*
| $\oplus$ e	*unary operator*
| $e_1 \otimes e_2$	*binary operator*
| **let** x=e_1 **in** e_2	*let*
| $\{e_2 \mid x \in e_1\}$	*comprehension*
| e_1 ? e_2 : e_3	*conditional*