

# 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 Aggregating Matching Patterns

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

# Calculus for Aggregating Matching Patterns

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

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

# Rules

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

*Evaluate p against each WME*  
*Evaluate p once*  
*Ensure p does hold for any WME*  
*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

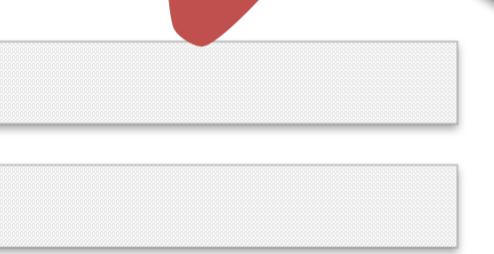
$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

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

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

JRules  
Engine



CAMP semantics

## Unary Operators

$\oplus d ::=$   
 $\mid \text{identity } d$  no-op. returns  $d$   
 $\mid \neg d$  negates a Boolean  
 $\mid \{d\}$  singleton bag of  $d$   
 $\mid \#d$  size of bag  
 $\mid \text{flatten } d$  flatten a bag of bags  
 $\mid [A:d]$  record constructor  
 $\mid d.A$  field selection  
 $\mid d-A$  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

$q ::= d$	<i>constant data</i>
$\text{In}$	<i>context value</i>
${}^\oplus q$	<i>unary operator</i>
$q_1 \otimes q_2$	<i>binary operator</i>
$\chi\langle q_2 \rangle(q_1)$	<i>map</i>
$\sigma\langle q_2 \rangle(q_1)$	<i>select</i>
$q_1 \times q_2$	<i>cartesian product</i>
$\bowtie^d\langle q_2 \rangle(q_1)$	<i>dependent join</i>
$q_1 \sqcup \sqcup q_2$	<i>default</i>

# 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

$r ::= \text{when } p; r$   
 $| \text{global } p; r$   
 $| \text{not } p; r$   
 $| \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $| \oplus p$   
 $| p_1 \otimes p_2$   
 $| \text{map } p$   
 $| \text{assert } p$   
 $| p_1 || p_2$   
 $| \text{it} | \text{let } \text{it} = p_1 \text{ in } p_2$   
 $| \text{env} | \text{let } \text{env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

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

JRules  
Engine



## Unary Operators

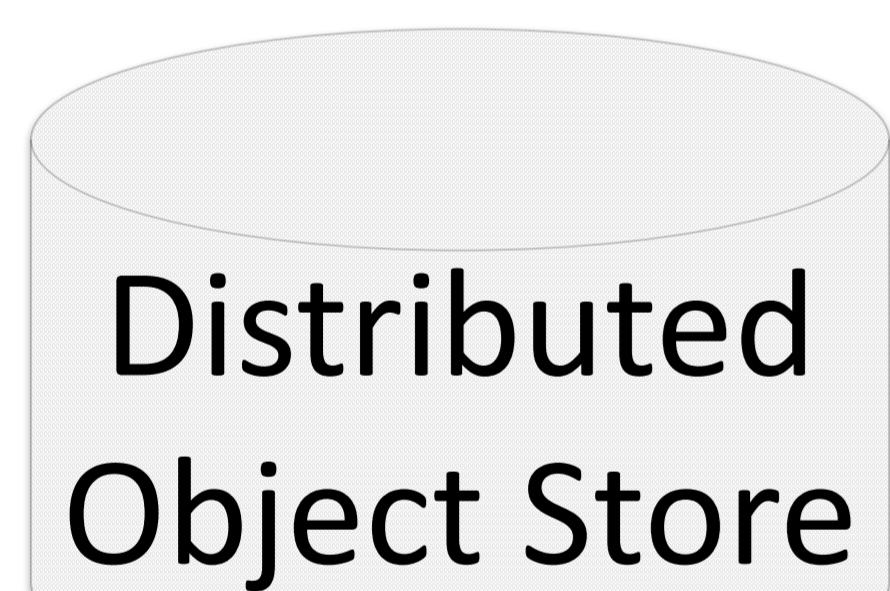
$\oplus d ::=$   
 $| \text{identity } d$  no-op. returns  $d$   
 $| \neg d$  negates a Boolean  
 $| \{d\}$  singleton bag of  $d$   
 $| \#d$  size of bag  
 $| \text{flatten } d$  flatten a bag of bags  
 $| [A:d]$  record constructor  
 $| d.A$  field selection  
 $| d-A$  field removal

$\oplus d \Downarrow_o d$

## 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

$d \otimes d \Downarrow_o d$



Distributed  
Object Store

Future  
Work

Nested Relational Algebra

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

$q @ d \Downarrow_a d$

# 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

$r ::= \text{when } p; r$   
 $| \text{global } p; r$   
 $| \text{not } p; r$   
 $| \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $| \oplus p$   
 $| p_1 \otimes p_2$   
 $| \text{map } p$   
 $| \text{assert } p$   
 $| p_1 || p_2$   
 $| \text{it} | \text{let } \text{it} = p_1 \text{ in } p_2$   
 $| \text{env} | \text{let } \text{env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

## Unary Operators

$\oplus d ::=$   
 $| \text{identity } d$  no-op. returns  $d$   
 $| \neg d$  negates a Boolean  
 $| \{d\}$  singleton bag of  $d$   
 $| \#d$  size of bag  
 $| \text{flatten } d$  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

$d \otimes d \Downarrow_o d$

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

[p]

?

Distributed Object Store

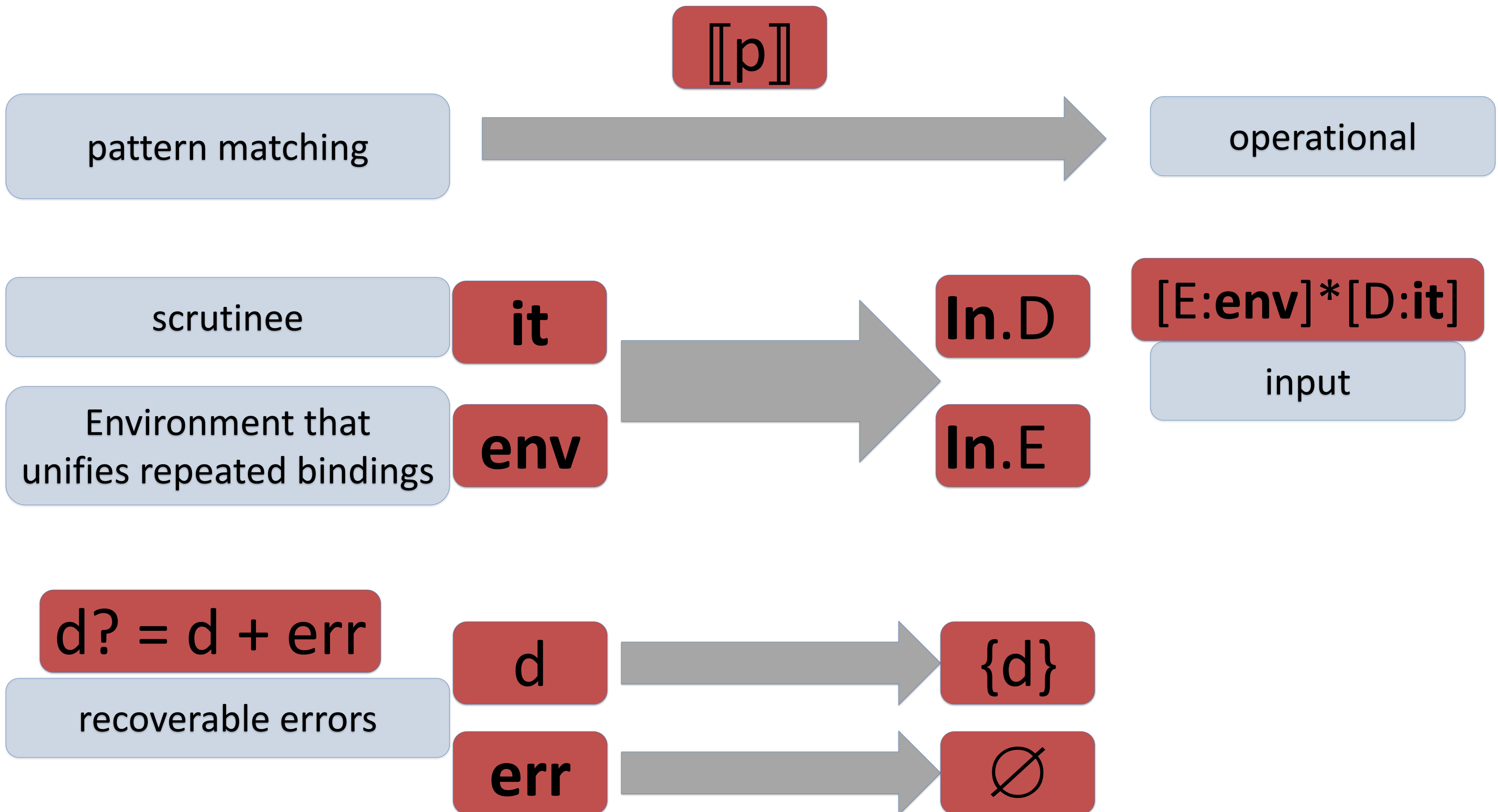
Future Work

## Nested Relational Algebra

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

$q @ d \Downarrow_a d$

# CAMP → NRA



# 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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $\mid \oplus p$   
 $\mid p_1 \otimes p_2$   
 $\mid \text{map } p$   
 $\mid \text{assert } p$   
 $\mid p_1 \parallel p_2$   
 $\mid \text{it} \mid \text{let } it = p_1 \text{ in } p_2$   
 $\mid \text{env} \mid \text{let env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

## Unary Operators

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

## 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

$d \otimes d \Downarrow_o d$

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

CAMP semantics

$\llbracket p \rrbracket ? @ [E:\sigma]^*[D:d_1] \Downarrow_a \{d_2\}$   
 $\llbracket p \rrbracket ? @ [E:\sigma]^*[D:d_1] \Downarrow_a \emptyset$   
 $\llbracket p \rrbracket ? @ [E:\sigma]^*[D:d_1] \Downarrow_a \text{err}$

## Compilation is Semantics Preserving

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

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



Future Work

## Nested Relational Algebra

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

# 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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

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

## Unary Operators

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

## 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

$d \otimes d \Downarrow_o d$

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

CAMP semantics

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow^a [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow^{\phi} \llbracket \sigma + p @ d_1 \Downarrow^a d_2 \rrbracket$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow^{\phi} \llbracket \sigma + p @ d_1 \Downarrow^{\phi} \text{err} \rrbracket$



Future Work

## Nested Relational Algebra

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

$d @ d \Downarrow_a p$

# N<sub>amed</sub> N<sub>ested</sub> R<sub>elational</sub> C<sub>alculus</sub>

e ::= x

*variables*

| d

*constant data*

|  $\oplus$  e

*unary operator*

|  $e_1 \otimes e_2$

*binary operator*

| **let** x=e<sub>1</sub> **in** e<sub>2</sub>

*let*

| {e<sub>2</sub> | x  $\in$  e<sub>1</sub>}

*comprehension*

| e<sub>1</sub> ? e<sub>2</sub> : e<sub>3</sub>

*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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

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

## Unary Operators

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

## Binary Operators

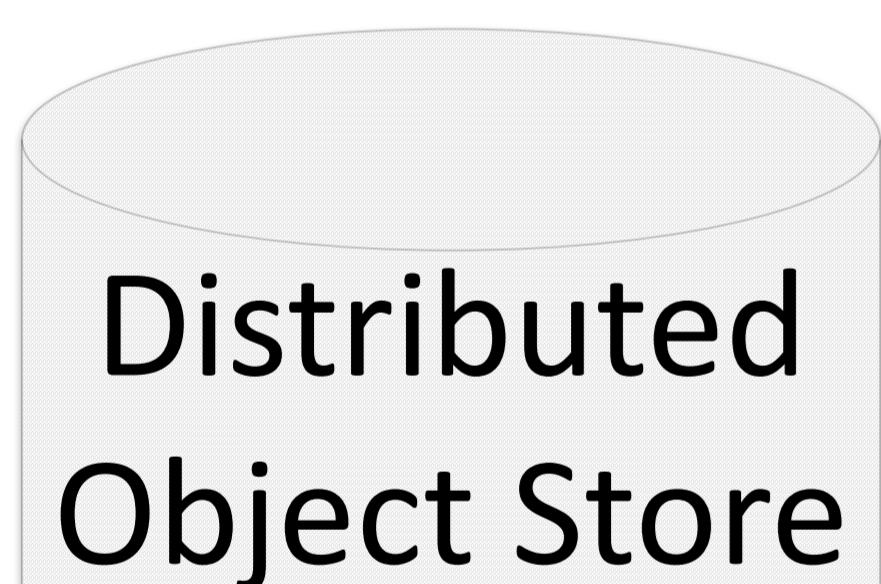
$d_1 \otimes d_2 ::=$   
 $\mid d_1 = d_2$  equality  
 $\mid d_1 \in d_2$  element of  
 $\mid d_1 \cup d_2$  union  
 $\mid d_1 * d_2$  biased record concat  
 $\mid d_1 + d_2$  compatible record concat

$d \otimes d \Downarrow_o d$

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

CAMP semantics

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\phi [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\diamond [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\circ [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\times [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\wedge [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\vee [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\exists [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\forall [d_2]$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\text{err}$



Future Work

## Nested Relational Algebra

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

$q @ d \Downarrow_a p$

$[q] \times ?$

$e @ d \Downarrow_o e$

## Named Nested Relational Calculus

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

# NRA → NNRC

$\llbracket q \rrbracket_x$

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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

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

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

## Unary Operators

$\oplus d ::=$	
$\mid \text{identity } d$	no-op. returns $d$
$\mid \neg d$	negates a Boolean
$\mid \{d\}$	singleton bag of $d$
$\mid \#d$	size of bag
$\mid \text{flatten } d$	flatten a bag of bags
$\mid [A:d]$	record constructor
$\mid d.A$	field selection
$\mid d-A$	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

$d \otimes d \Downarrow_o d$

## Compilation is Semantics Preserving

if  $\sigma(x) = d_1$

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



Future Work

**Nested Relational Algebra**

$q ::= d$	constant data
$\mid \text{In}$	context value
$\mid \oplus q$	unary operator
$\mid q_1 \otimes q_2$	binary operator
$\mid x(q_2)(q_1)$	map
$\mid \sigma(q_2)(q_1)$	select
$\mid q_1 \times q_2$	cartesian product
$\mid \bowtie^d(q_2)(q_1)$	dependent join
$\mid q_1 \parallel q_2$	default

$d @ d \Downarrow_a d$

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
$\mid d$	constant data
$\mid \oplus e$	unary operator
$\mid e_1 \otimes e_2$	binary operator
$\mid \text{let } x=e_1 \text{ in } e_2 \text{ let}$	
$\mid \{e_2 \mid x \in e_1\}$	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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $\mid \oplus p$   
 $\mid p_1 \otimes p_2$   
 $\mid \text{map } p$   
 $\mid \text{assert } p$   
 $\mid p_1 \parallel p_2$   
 $\mid \text{it} \mid \text{let } \text{it} = p_1 \text{ in } p_2$   
 $\mid \text{env} \mid \text{let } \text{env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

## Unary Operators

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

## 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

$d \otimes d \Downarrow_o d$

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

CAMP semantics

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a d_2$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_2] \Downarrow_a d_1$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_\phi \sigma + p @ d_1 \Downarrow_o d_2$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_2] \Downarrow_\phi \sigma + p @ d_2 \Downarrow_o d_1$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1, err] \Downarrow_\phi \sigma + p @ d_1 \Downarrow_o d_2$



Future Work

## Nested Relational Algebra

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

$d @ d \Downarrow_a p$

Tell?

if  $\sigma(x) = d_1$   
 $q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d_2$

## Named Nested Relational Calculus

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

# NNRC → CAMP

[e]

declarative

pattern matching

environment

x

renaming

env.x

scrutinee

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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $\mid \oplus p$   
 $\mid p_1 \otimes p_2$   
 $\mid \text{map } p$   
 $\mid \text{assert } p$   
 $\mid p_1 \parallel p_2$   
 $\mid \text{it} \mid \text{let } it = p_1 \text{ in } p_2$   
 $\mid \text{env} \mid \text{let env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

## Unary Operators

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

## 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

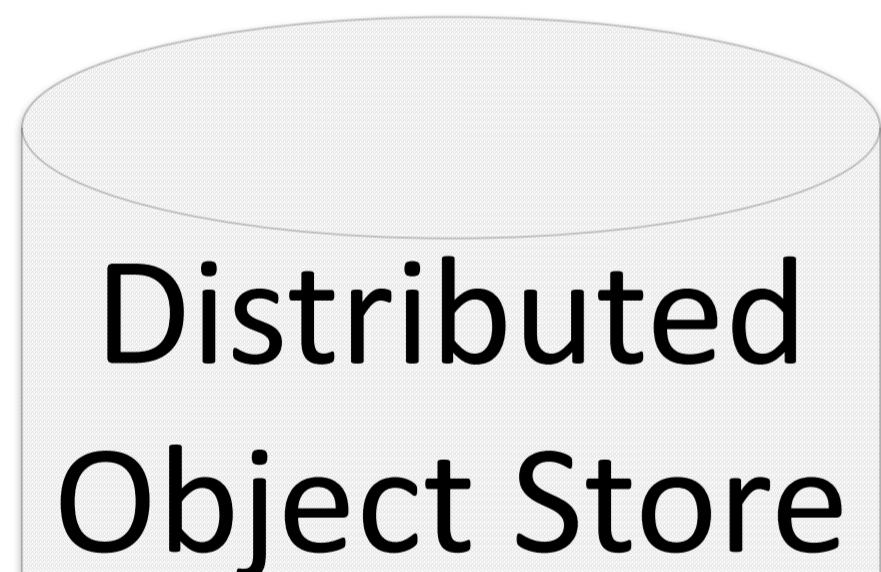
$d \otimes d \Downarrow_o d$

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

$\oplus d \Downarrow_o d$

CAMP semantics

$\sigma \vdash [e]@ [E:\sigma]^*[D:d_1] \Downarrow_o [d_2]$   
 $\sigma \vdash [e]@ [E:\sigma]^*[D:d_2] \Downarrow_o [d_1]$   
 $\sigma \vdash p @ d_1 \Downarrow_r d_2$   
 $\sigma \vdash p @ d_2 \Downarrow_r d_1$   
 $\sigma \vdash e @ d_0 \Downarrow_r d$   
 $\sigma \vdash e @ d_0 \Downarrow_r d$



Distributed  
Object Store

## Compilation is Semantics Preserving

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

$| \bowtie^d (q_2)(q_1)$  dependent join  
 $| q_1 \parallel q_2$  default

**Named Nested Relational Calculus**

$e ::= x$	variables
$\mid d$	constant data
$\mid \oplus e$	unary operator
$\mid e_1 \otimes e_2$	binary operator
$\mid \text{let } x=e_1 \text{ in } e_2 \text{ let}$	
$\mid \{e_2 \mid x \in e_1\}$	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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

$p ::= d$   
 $\mid \oplus p$   
 $\mid p_1 \otimes p_2$   
 $\mid \text{map } p$   
 $\mid \text{assert } p$   
 $\mid p_1 \parallel p_2$   
 $\mid \text{it} \mid \text{let } it = p_1 \text{ in } p_2$   
 $\mid \text{env} \mid \text{let env} += p_1 \text{ in } p_2$

*constant data*  
*unary operator*  
*binary operator*  
*map over a bag*  
*assertion*  
*error recovery (orElse)*  
*get/set scrutinee*  
*get/update environment*

## Unary Operators

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

## 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

$d \otimes d \Downarrow_o d$

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

Compilers  
 $\llbracket p \rrbracket$ ,  $\llbracket q \rrbracket_x$ , and  $\llbracket e \rrbracket$   
produce code at  
most a constant  
times larger than  
their input

$$q @ d \Downarrow_a p \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d \leftrightarrow \text{if } \sigma(x) = d \text{ then } p \text{ else } \llbracket q \rrbracket_x \Downarrow_c d$$



Future Work

**Nested Relational Algebra**

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

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket d_1 \rrbracket$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket d_1 \rrbracket \Downarrow_a \llbracket d_2 \rrbracket$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket d_1 \rrbracket \Downarrow_a \llbracket d_2 \rrbracket \Downarrow_a \llbracket d \rrbracket$   
 $\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a \llbracket d_1 \rrbracket \Downarrow_a \llbracket d_2 \rrbracket \Downarrow_a \llbracket d \rrbracket \Downarrow_a \llbracket \text{err} \rrbracket$

$\sigma \vdash \llbracket e \rrbracket @ d_0 \Downarrow_r d$   
 $\sigma \vdash e \Downarrow_c d$

**Named Nested Relational Calculus**

$e ::= x$	variables
$\mid d$	constant data
$\mid \oplus e$	unary operator
$\mid e_1 \otimes e_2$	binary operator
$\mid \text{let } x = e_1 \text{ in } e_2 \text{ let}$	
$\mid \{e_2 \mid x \in e_1\}$	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



## IBM JRules

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



# A Pattern Calculus for Rule Languages: Semantics, Compilation, and Mechanization

Yann Shinnar, Jérôme Siméon, and Martin Hirzel

IBM Research



Everything in  
gray has been  
verified in Coq



## Type Soundness

$$\sigma :_d \Gamma$$

$$d_0 :_d \tau_0$$

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

*implies that*  $\exists d_1?$ ,

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

$$d_1? :_d \tau_1$$

Futu  
Wo

### Nested Relational Algebra

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

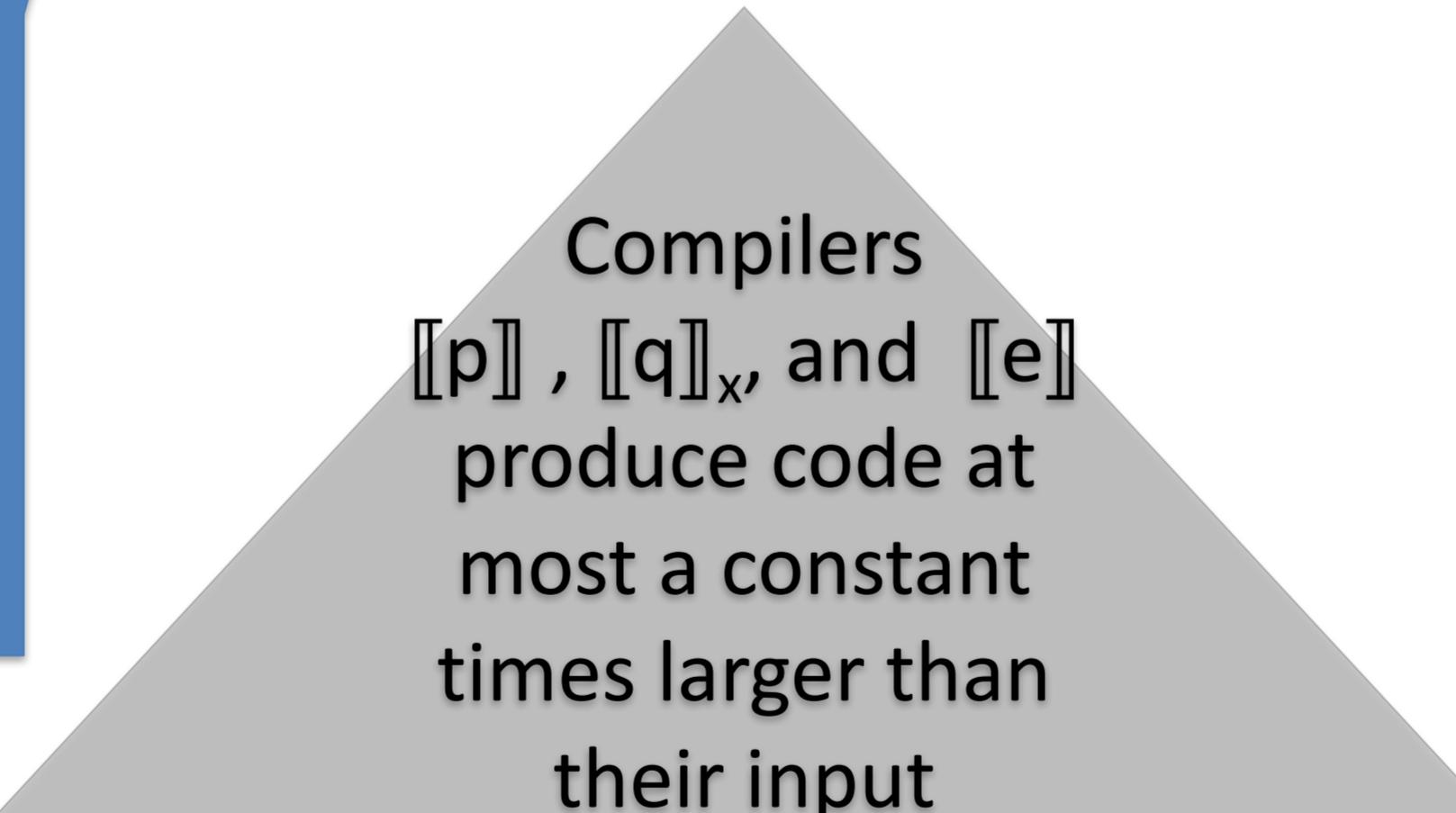
$$d @ d \Downarrow_a d$$

Calculus for Aggregating Matching Patterns	
$  p$	constant data
$  p_1 \otimes p_2$	unary operator
$  \text{it} = p_1 \text{ in } p_2$	binary operator
$  \text{et env} += p_1 \text{ in } p_2$	map over a bag
$  \text{assert}$	assertion
$  \text{error recovery (orElse)}$	error recovery (orElse)
$  \text{get/set scrutinee}$	get/set scrutinee
$  \text{get/update environment}$	get/update environment

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

$$\begin{aligned} &\sigma :_d \Gamma \\ &d_0 :_d \tau_0 \\ &\Gamma \vdash p :_r \tau_0 \rightarrow \tau_1 \\ &\text{implies that } \exists d_1?, \\ &\sigma \vdash p @ d_0 \Downarrow_r d_1? \\ &d_1? :_d \tau_1 \end{aligned}$$

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



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

### Unary Operators

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

$$\oplus d \Downarrow_o d$$

$$\begin{aligned} &d_0 :_d \tau_0 \\ &\oplus :_o \tau_0 \rightarrow \tau_1 \\ &\text{implies that } \exists d_1, \\ &\oplus d_0 \Downarrow_o d_1 \\ &d_1 :_d \tau_1 \end{aligned}$$

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

### 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

$$d \otimes d \Downarrow_o d$$

$$\begin{aligned} &d_0 :_d \tau_0 \\ &d_1 :_d \tau_1 \\ &\otimes :_o \tau_0 \rightarrow \tau_1 \rightarrow \tau_2 \\ &\text{implies that } \exists d_2, \\ &d_0 \otimes d_1 \Downarrow_o d_2 \\ &d_2 :_d \tau_2 \end{aligned}$$

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

$$d @ d \Downarrow_a d \leftrightarrow \sigma \vdash e \Downarrow_c d$$

Named Nested Relational Calculus	
$e ::= x$	variables
$  d$	constant data
$  \oplus e$	unary operator
$  e_1 \otimes e_2$	binary operator
$  \text{let } x = e_1 \text{ in } e_2 \text{ 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



## 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

$r ::= \text{when } p; r$   
 $\mid \text{global } p; r$   
 $\mid \text{not } p; r$   
 $\mid \text{return } p$

## Calculus for Aggregating Matching Patterns

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

## Unary Operators

$\oplus d ::=$	
$\mid \text{identity } d$	<i>no-op. returns d</i>
$\mid \neg d$	<i>negates a Boolean</i>
$\mid \{d\}$	<i>singleton bag of d</i>
$\mid \#d$	<i>size of bag</i>
$\mid \text{flatten } d$	<i>flatten a bag of bags</i>
$\mid [A:d]$	<i>record constructor</i>
$\mid d.A$	<i>field selection</i>
$\mid d-A$	<i>field removal</i>

## Binary Operators

$d_1 \otimes d_2 ::=$	
$\mid d_1 = d_2$	<i>equality</i>
$\mid d_1 \in d_2$	<i>element of</i>
$\mid d_1 \cup d_2$	<i>union</i>
$\mid d_1 * d_2$	<i>biased record concat</i>
$\mid d_1 + d_2$	<i>compatible record concat</i>

$d \otimes d \Downarrow_o d$

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

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

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

$\Gamma \vdash e :_c \tau$

$q :_a \tau_0 \rightarrow \tau_1$



Distributed  
Object Store

**Nested Relational Algebra**

$q ::= d$	<i>constant data</i>
$\mid \text{In}$	<i>context value</i>
$\mid \oplus q$	<i>unary operator</i>
$\mid q_1 \otimes q_2$	<i>binary operator</i>
$\mid \chi(q_2)(q_1)$	<i>map</i>
$\mid \sigma(q_2)(q_1)$	<i>select</i>
$\mid q_1 \times q_2$	<i>cartesian product</i>
$\mid \bowtie^d(q_2)(q_1)$	<i>dependent join</i>
$\mid q_1 \parallel q_2$	<i>default</i>

$\llbracket p \rrbracket @ \llbracket E \rrbracket$   
 $\llbracket p \rrbracket_x @ \llbracket V \rrbracket$

$d @ d \Downarrow_a p$

Compilers  
 $\llbracket p \rrbracket, \llbracket q \rrbracket_x,$  and  $\llbracket e \rrbracket$   
 produce code at  
 most a constant  
 times larger than  
 their input

$if \sigma(x) = d_1$   
 $q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d_2$

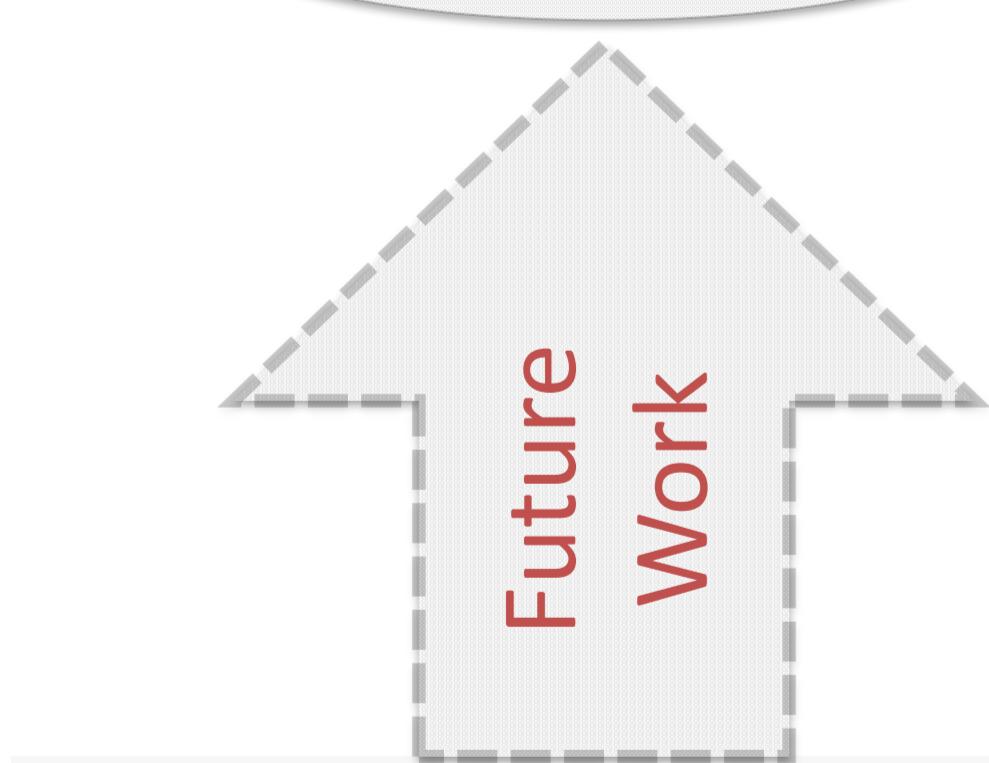
**Named Nested Relational Calculus**

$e ::= x$	<i>variables</i>
$\mid d$	<i>constant data</i>
$\mid \oplus e$	<i>unary operator</i>
$\mid e_1 \otimes e_2$	<i>binary operator</i>
$\mid \text{let } x = e_1 \text{ in } e_2 \text{ let}$	
$\mid \{e_2 \mid x \in e_1\}$	<i>comprehension</i>
$\mid e_1 ? e_2 : e_3$	<i>conditional</i>

# Type Soundness

$$\begin{array}{l} d_0 :_d \tau_0 \\ q :_a \tau_0 \rightarrow \tau_1 \\ \text{implies that } \exists d_1, \\ q @ d_0 \Downarrow_a d_1 \\ d_1 :_d \tau_1 \end{array}$$

Distributed Object Store



Nested Relational Algebra	
$q ::= d$	constant data
$  \text{In}$	context value
$  \oplus q$	unary operator
$  q_1 \otimes q_2$	binary operator
$  \chi(q_2)(q_1)$	map
$  \sigma(q_2)(q_1)$	select
$  q_1 \times q_2$	cartesian product
$  \bowtie^d(q_2)(q_1)$	dependent join
$  q_1    q_2$	default

Named Nested Relational Calculus	
$e ::= x$	variables
$  d$	constant data
$  \oplus e$	unary operator
$  e_1 \otimes e_2$	binary operator
$  \text{let } x = e_1 \text{ in } e_2 \text{ let}$	
$  \{e_2 \mid x \in e_1\}$	comprehension
$  e_1 ? e_2 : e_3$	conditional



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

- constant data
- unary operator
- binary operator
- map over a bag
- assertion
- error recovery (optional)
- get/set scrutinee
- get/update environment

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

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

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

# Type Soundness

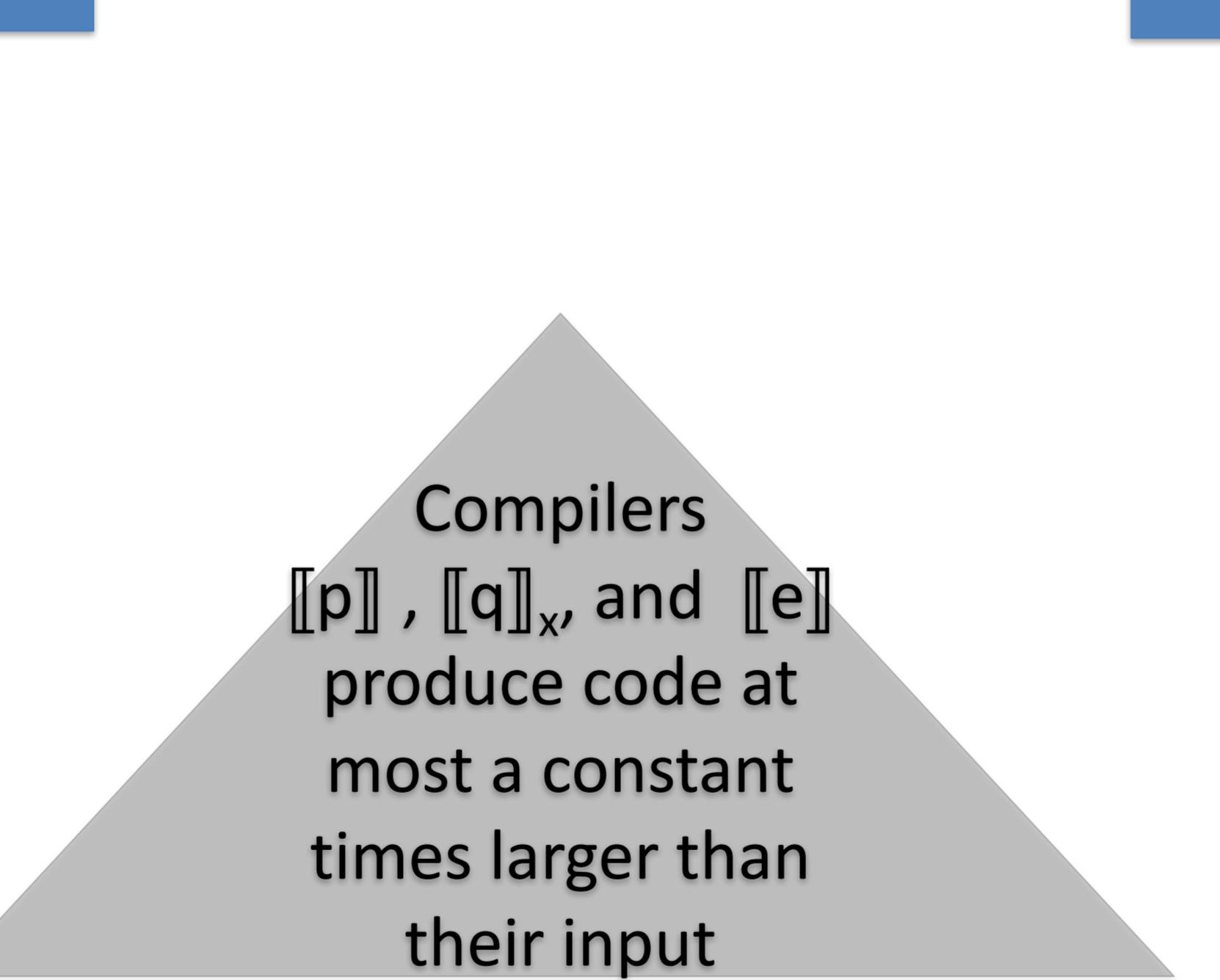
$$\sigma :_d \Gamma$$

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

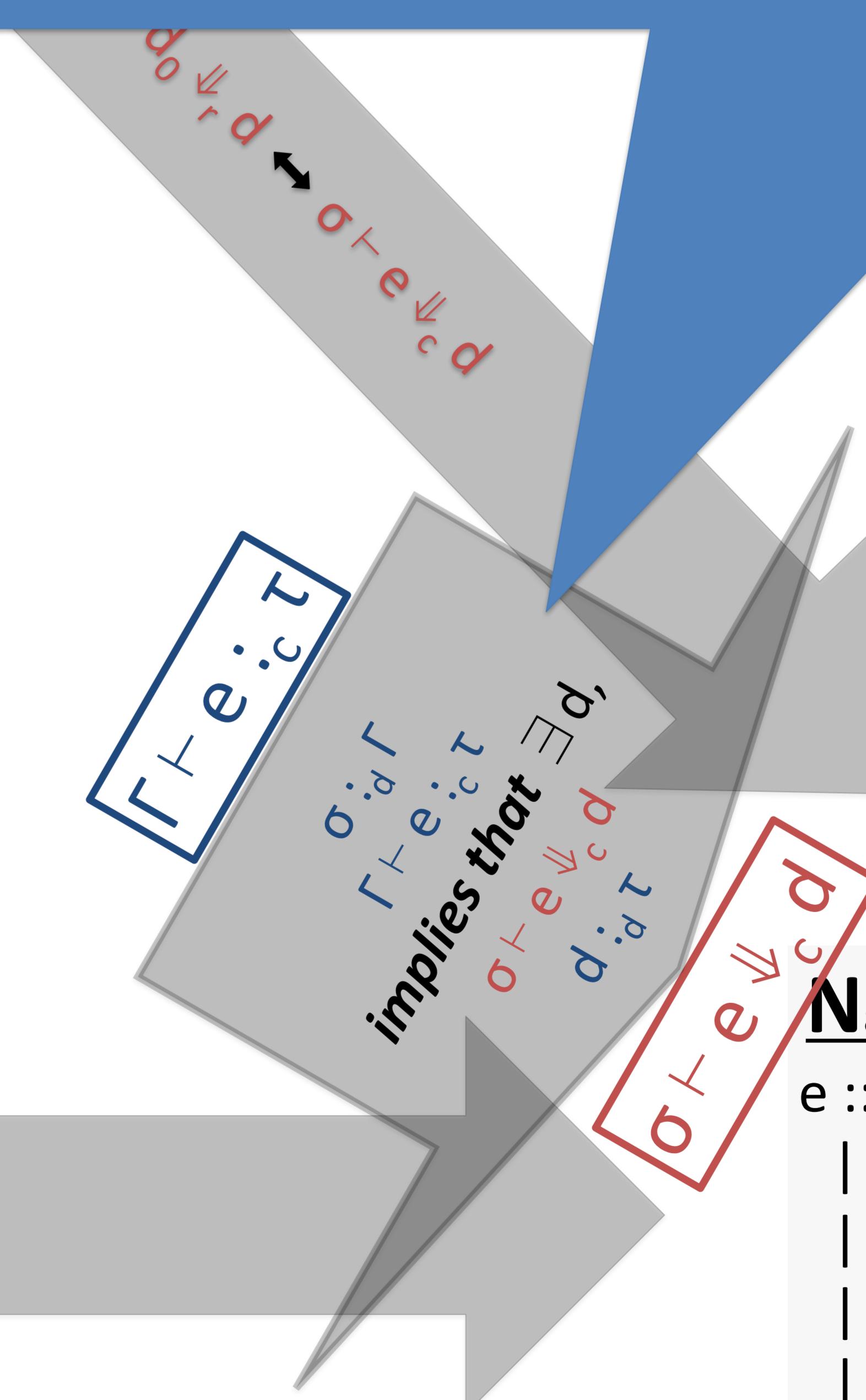
*implies that*  $\exists d,$

$$\sigma \vdash e \Downarrow_c d$$

$$d :_d \tau$$



$$\text{if } \sigma(x) = d_1 \\ q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d_2$$





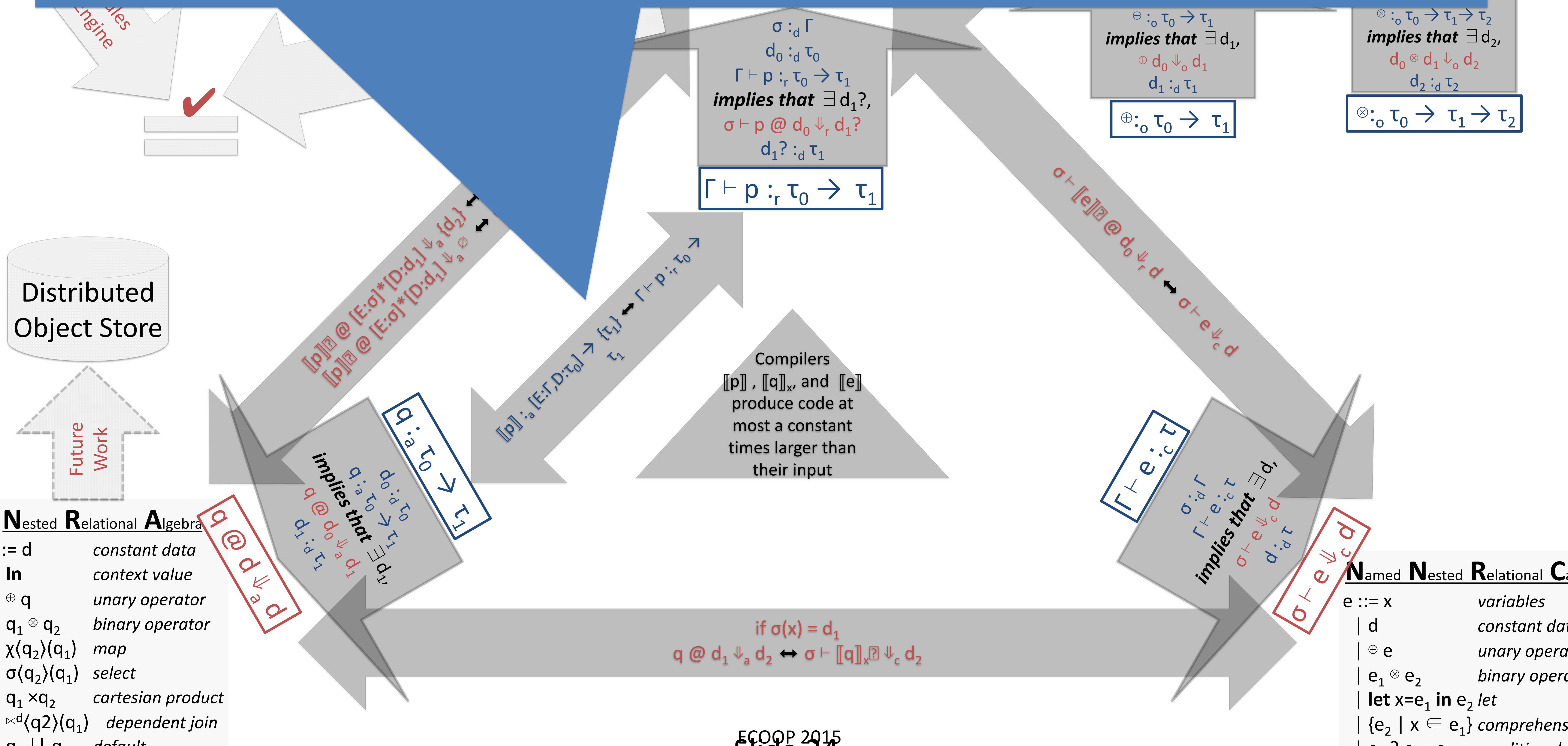
## Type Preservation

$$\llbracket p \rrbracket :_a [E:\Gamma, D:\tau_0] \rightarrow \{\tau_1\} \leftrightarrow \Gamma \vdash p :_r \tau_0 \rightarrow \tau_1$$

```

rule First
where
C:
Ms
A:
} do
} the
inst
}

```



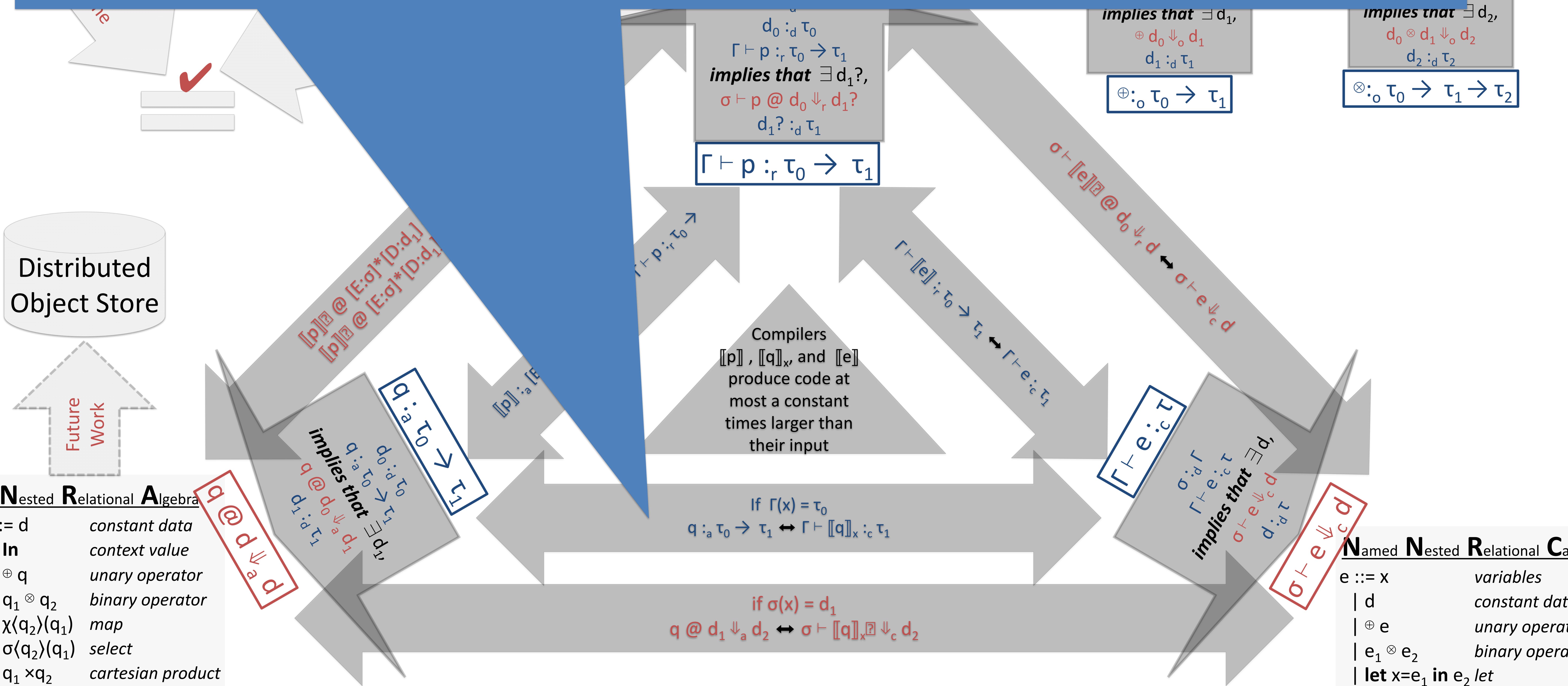
$$q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d_2$$



# Type Preservation

If  $\Gamma(x) = \tau_0$

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





Everything in gray has been verified in Coq

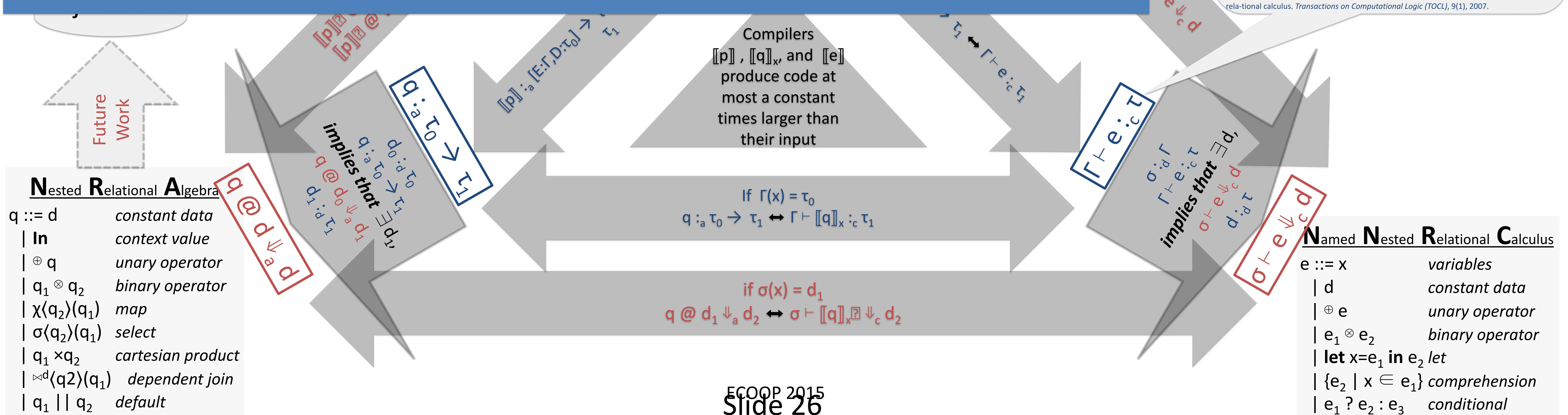


## Polymorphic type inference for NNRC

- 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. Vansumeren. Polymorphic type inference for the named nested relational calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.



# 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



## 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

$$r ::= \text{when } p; r \mid \text{global } p; r \mid \text{not } p; r \mid \text{return } p$$

## Calculus for Aggregating Matching Patterns

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

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

$\oplus d \Downarrow_o d$

$d_0 :_d \tau_0$

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

**implies that**  $\exists d_1,$

$\oplus d_0 \Downarrow_o d_1$

$d_1 :_d \tau_1$

## 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

$d \otimes d \Downarrow_o d$

$d_0 :_d \tau_0$

$d_1 :_d \tau_1$

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

**implies that**  $\exists d_2,$

$d_0 \otimes d_1 \Downarrow_o d_2$

$d_2 :_d \tau_2$

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



Future Work

## Nested Relational Algebra

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

$d @ d \Downarrow_a p$

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a [d_2]$

$\llbracket p \rrbracket @ [E:\sigma]^*[D:d_1] \Downarrow_a [d_2]$

Compilers  
 $\llbracket p \rrbracket, \llbracket q \rrbracket_x, \text{ and } \llbracket e \rrbracket$   
produce code at  
most a constant  
times larger than  
their input

If  $\Gamma(x) = \tau_0$   
 $q :_a \tau_0 \rightarrow \tau_1 \leftrightarrow \Gamma \vdash \llbracket q \rrbracket_x :_c \tau_1$

if  $\sigma(x) = d_1$   
 $q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash \llbracket q \rrbracket_x \Downarrow_c d_2$

ECOOP 2015

## 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. Vansumeren. Polymorphic type inference for the named nested relational calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.

## Named Nested Relational Calculus

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