

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

Avraham Shinnar, Jérôme Siméon, and

Martin Hirzel

IBM Research

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

$p ::= d$	<i>constant data</i>
$\oplus p$	<i>unary operator</i>
$p_1 \otimes p_2$	<i>binary operator</i>
map p	<i>map over a bag</i>
assert p	<i>assertion</i>
$p_1 p_2$	<i>error recovery (orElse)</i>
it let it = p₁ in p ₂	<i>get/set scrutinee</i>
env let env += p₁ in p ₂	<i>get/update environment</i>

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

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

```
it.type = "Marketer"  
 $\wedge$  env.C.data.id  $\in$  it.data.clients  
 $\wedge$  let env += [M : it] in env
```

Rules

$r ::= \mathbf{when} \ p; \ r$

| $\mathbf{global} \ p; \ r$

| $\mathbf{not} \ p; \ r$

| $\mathbf{return} \ p$

Evaluate p against each WME

Evaluate p once

Ensure p does hold for any WME

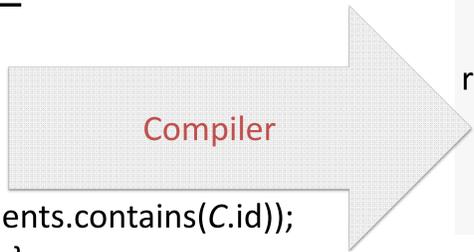
Compute a result using p

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);
  }
}
```

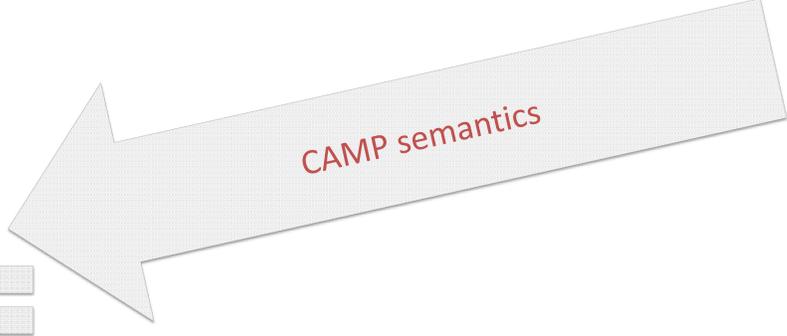


Rules
 $r ::= \text{when } p; r$
 | **global** $p; r$
 | **not** $p; r$
 | **return** p

Calculus for Aggregating Matching Patterns
 $p ::= d$
 | $\oplus p$ *constant data*
 | $\oplus p$ *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
 $\oplus d ::=$
 | **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*



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

$\oplus d \Downarrow_o d$

$d \otimes d \Downarrow_o d$

Nested Relational Algebra

$q ::= d$	<i>constant data</i>
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>
$q_1 q_2$	<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 ::= when p; r
| global p; r
| not p; r
| return p
```

Calculus for Aggregating Matching Patterns

```
p ::= d
| ⊕ p
| p1 ⊗ p2
| map p
| assert p
| p1 || p2
| it | let it = p1 in p2
| env | let env += p1 in p2
| constant data
| unary operator
| binary operator
| map over a bag
| assertion
| error recovery (orElse)
| get/set scrutinee
| get/update environment
```

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

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

```
d1 ⊗ d2 ::=
| d1 = d2 equality
| d1 ∈ d2 element of
| d1 ∪ d2 union
| d1 * d2 biased record concat
| d1 + d2 compatible record concat
```

$d \otimes d \Downarrow_o d$

CAMP semantics

JRules Engine

Distributed Object Store

Future Work

Nested Relational Algebra

```
q ::= d
| 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
```

$q @ d \Downarrow_a q$

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
| ⊕ p
| p1 ⊗ p2
| map p
| assert p
| p1 || p2
| it | let it = p1 in p2
| env | let env += p1 in p2
| constant data
| unary operator
| binary operator
| map over a bag
| assertion
| error recovery (orElse)
| get/set scrutinee
| get/update environment
```

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

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

```
d1 ⊗ d2 ::=
| d1 = d2 equality
| d1 ∈ d2 element of
| d1 ∪ d2 union
| d1 * d2 biased record concat
| d1 + d2 compatible record concat
```

$d \otimes d \Downarrow_o d$

$[p]$

Distributed Object Store

Future Work

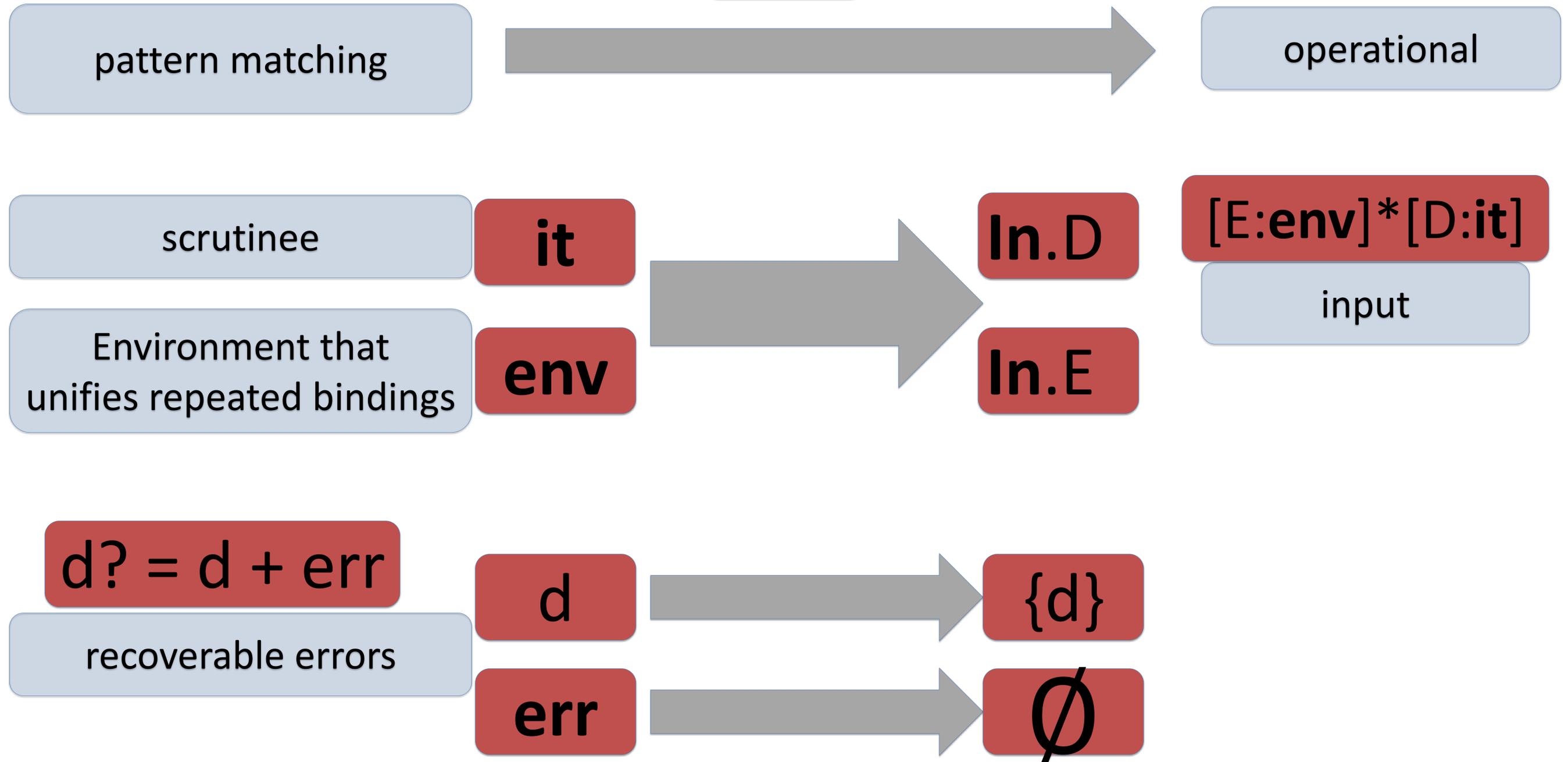
Nested Relational Algebra

```
q ::= d
| 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
```

$q @ d \Downarrow_a d$

CAMP → NRA

$[[p]]$



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

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

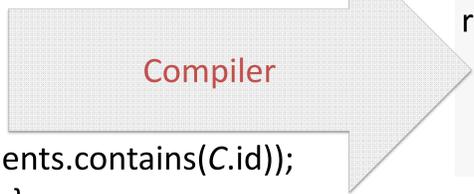


Everything in gray has been verified in Coq



IBM JRules

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

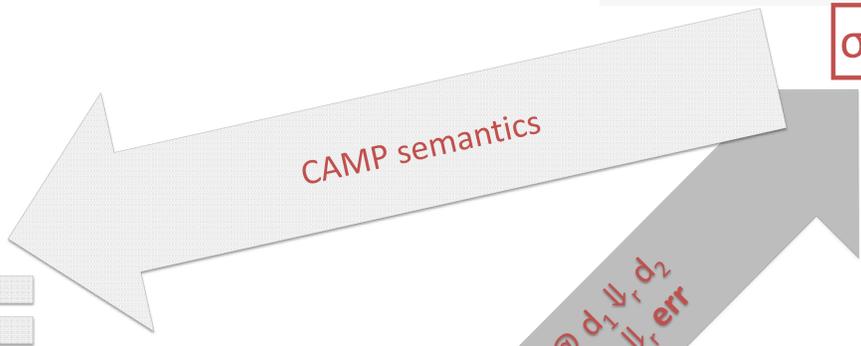


Rules
 $r ::=$ when p ; r
 | global p ; r
 | not p ; r
 | return p

Calculus for Aggregating Matching Patterns
 $p ::=$ d
 | $\oplus p$ constant data
 | 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
 $\oplus d ::=$
 | 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

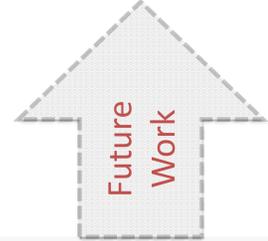
Binary Operators
 $d_1 \otimes d_2 ::=$
 | $d_1 = d_2$ equality
 | $d_1 \in d_2$ element of
 | $d_1 \cup d_2$ union
 | $d_1 * d_2$ biased record concat
 | $d_1 + d_2$ compatible record concat



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

$$\oplus d \Downarrow_o d$$

$$d \otimes d \Downarrow_o d$$



Nested Relational Algebra
 $q ::=$ d constant data
 | 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

$$p @ a$$

Compilation is Semantics Preserving

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

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

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



IBM JRules

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

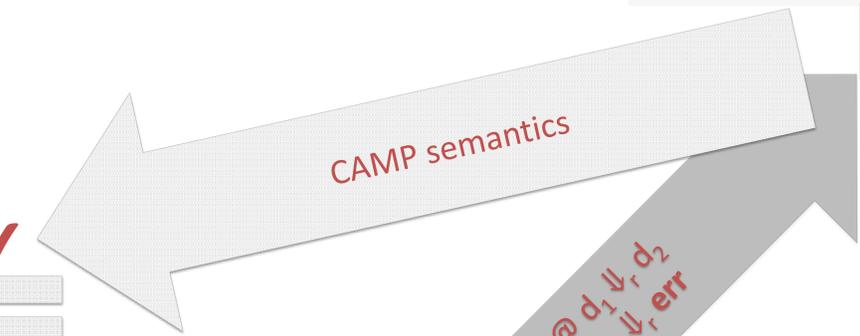


Rules
 $r ::= \text{when } p; r$
 $\text{global } p; r$
 $\text{not } p; r$
 $\text{return } p$

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

Unary Operators
 $\oplus d ::=$
 $\text{identity } d$ *no-op. returns d*
 $-d$ *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*

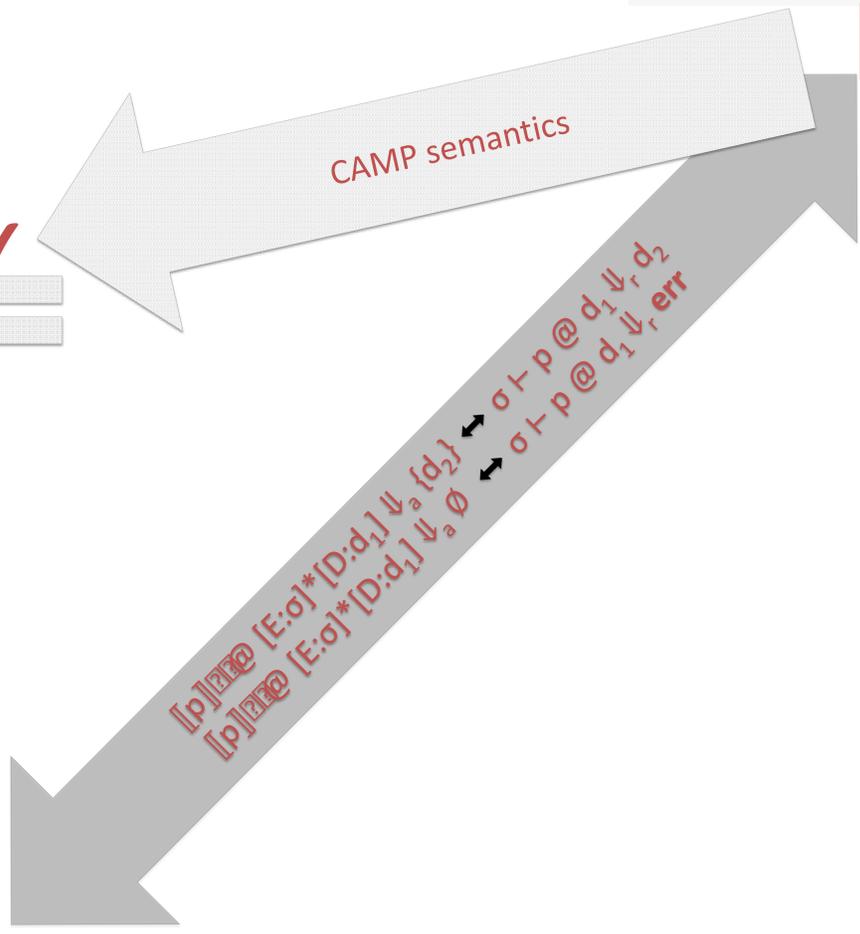
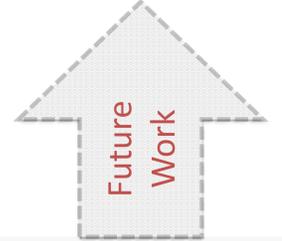
Binary Operators
 $d_1 \otimes d_2 ::=$
 $d_1 = d_2$ *equality*
 $d_1 \in d_2$ *element of*
 $d_1 \cup d_2$ *union*
 $d_1 * d_2$ *biased record concat*
 $d_1 + d_2$ *compatible record concat*



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

$\oplus d \Downarrow_o d$

$d \otimes d \Downarrow_o d$



Nested Relational Algebra
 $q ::= d$ *constant data*
 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*
 $q_1 \parallel q_2$ *default*

$q @ d \Downarrow_a d$

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

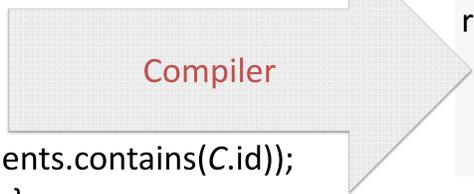


Everything in gray has been verified in Coq



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
      ⊕ p
      p1 ⊗ p2
      map p
      assert p
      p1 || p2
      it | let it = p1 in p2
      env | let env += p1 in p2
```

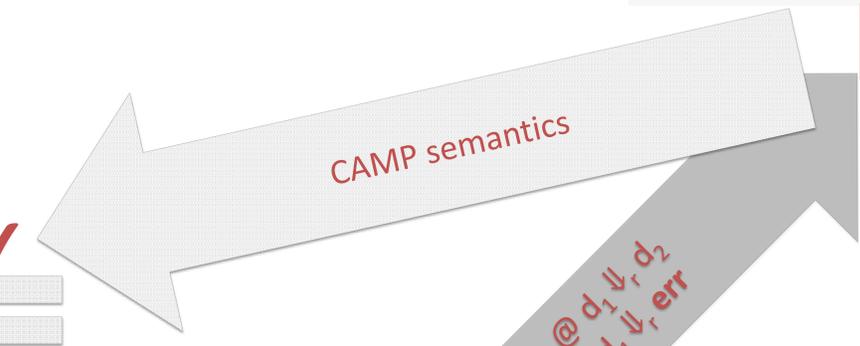
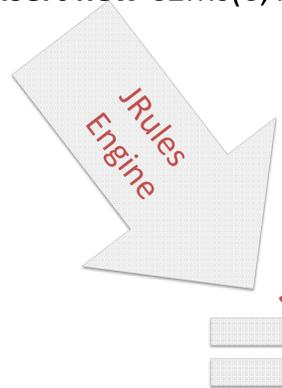
constant data
unary operator
binary operator
map over a bag
assertion
error recovery (orElse)
get/set scrutinee
get/update environment

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

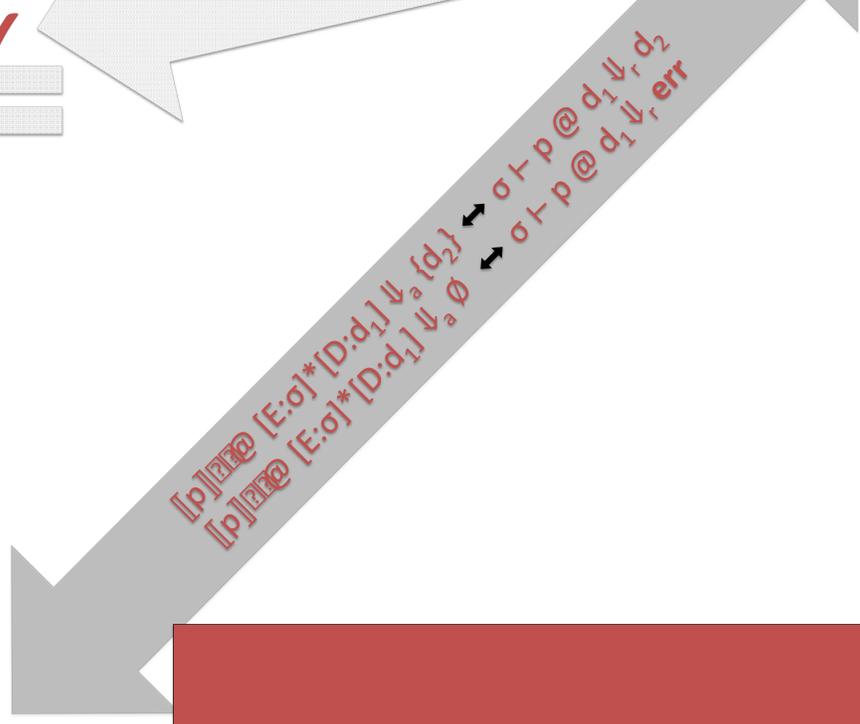
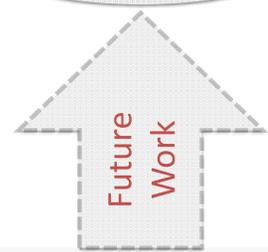
```
d1 ⊗ d2 ::=
  | d1 = d2 equality
  | d1 ∈ d2 element of
  | d1 ∪ d2 union
  | d1 * d2 biased record concat
  | d1 + d2 compatible record concat
```



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

$$\oplus d \Downarrow_o d$$

$$d \otimes d \Downarrow_o d$$

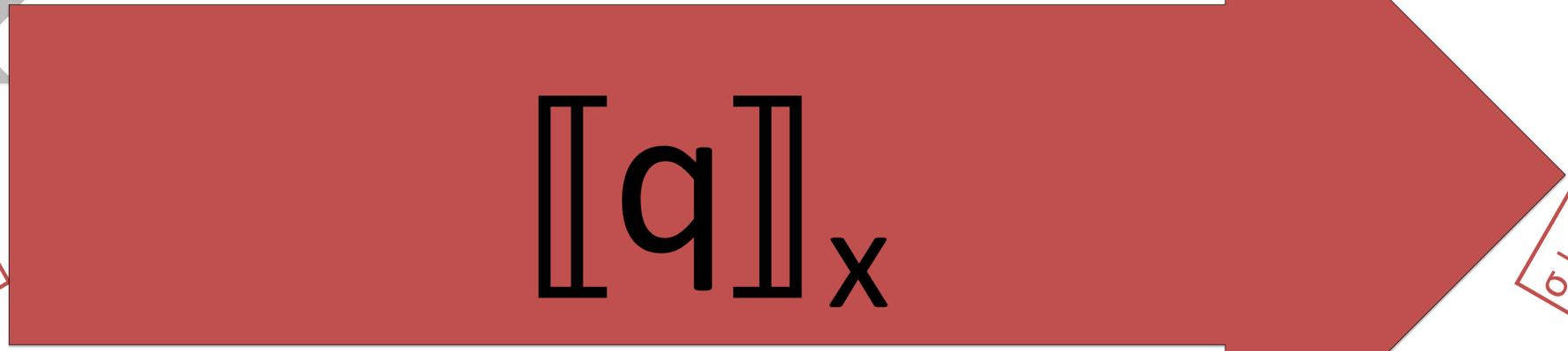


Nested Relational Algebra

```
q ::= d
      In
      ⊕ q
      q1 ⊗ q2
      χ(q2)(q1)
      σ(q2)(q1)
      q1 × q2
      ⋈d(q2)(q1)
      q1 || q2
```

constant data
context value
unary operator
binary operator
map
select
cartesian product
dependent join
default

$$q @ d \Downarrow_a d$$



$$[q]_x$$

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

Named Nested Relational Calculus

```
e ::= x
      d
      ⊕ e
      e1 ⊗ e2
      let x=e1 in e2
      {e2 | x ∈ e1}
      e1 ? e2 : e3
```

variables
constant data
unary operator
binary operator
let
comprehension
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

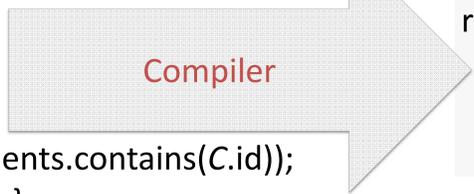


Everything in gray has been verified in Coq



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
| ⊕ p
| p1 ⊗ p2
| map p
| assert p
| p1 || p2
| it | let it = p1 in p2
| env | let env += p1 in p2
| constant data
| unary operator
| binary operator
| map over a bag
| assertion
| error recovery (orElse)
| get/set scrutinee
| 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

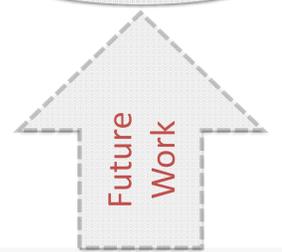
```
d1 ⊗ d2 ::=
| d1 = d2 equality
| d1 ∈ d2 element of
| d1 ∪ d2 union
| d1 * d2 biased record concat
| d1 + d2 compatible record concat
```



Compilation is Semantics Preserving

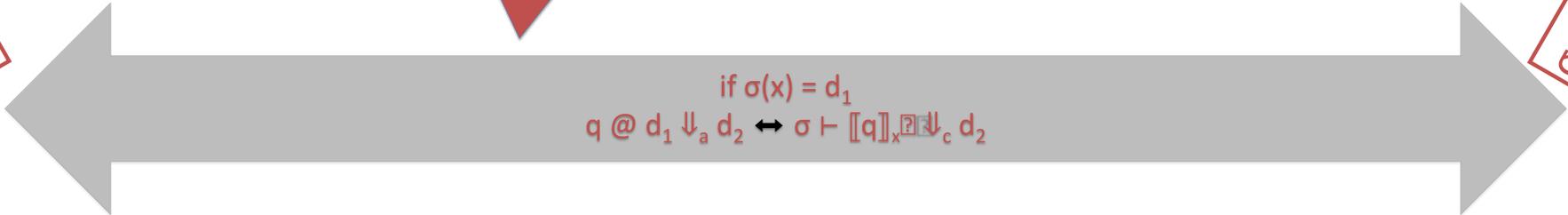
if $\sigma(x) = d_1$

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



Nested Relational Algebra

```
q ::= d
| 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
```



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



Named Nested Relational Calculus

```
e ::= x
| d
| ⊕ e
| e1 ⊗ e2
| let x=e1 in e2
| {e2 | x ∈ e1}
| e1 ? e2 : e3
| variables
| constant data
| unary operator
| binary operator
| let
| comprehension
| conditional
```

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

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

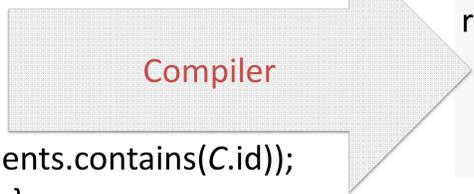


Everything in gray has been verified in Coq



IBM JRules

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



Rules

```
r ::= when p; r
      global p; r
      not p; r
      return p
```

Calculus for Aggregating Matching Patterns

```
p ::= d
      ⊕ p
      p1 ⊗ p2
      map p
      assert p
      p1 || p2
      it | let it = p1 in p2
      env | let env += p1 in p2
```

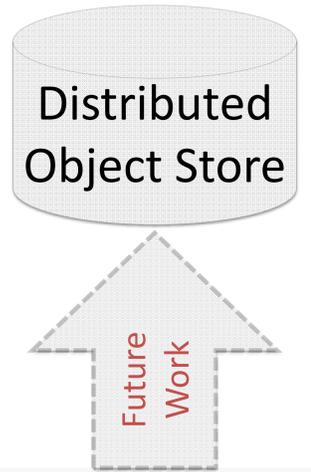
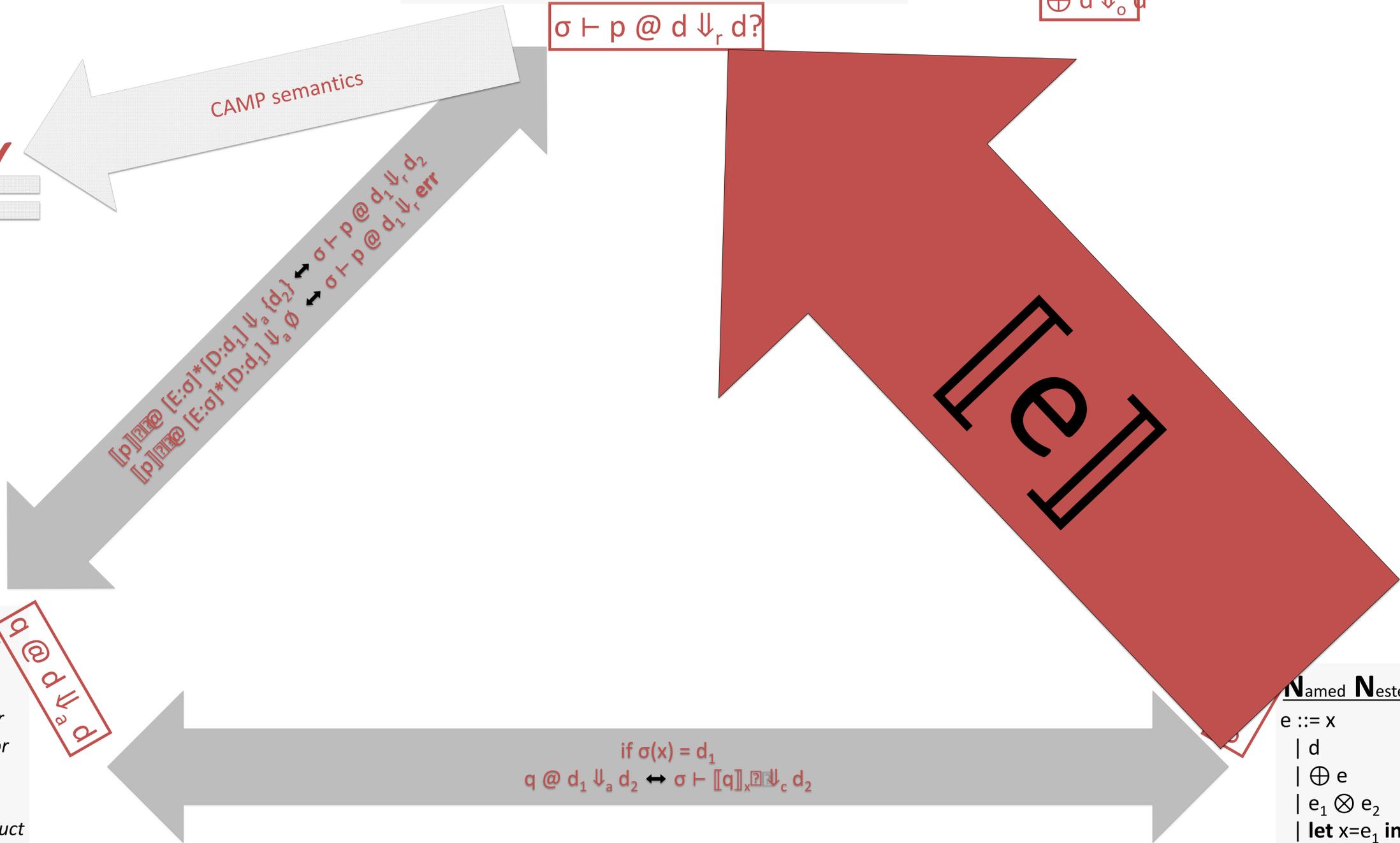
constant data
unary operator
binary operator
map over a bag
assertion
error recovery (orElse)
get/set scrutinee
get/update environment

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

```
d1 ⊗ d2 ::=
  | d1 = d2 equality
  | d1 ∈ d2 element of
  | d1 ∪ d2 union
  | d1 * d2 biased record concat
  | d1 + d2 compatible record concat
```



Nested Relational Algebra

```
q ::= d
      In
      ⊕ q
      q1 ⊗ q2
      χ(q2)(q1)
      σ(q2)(q1)
      q1 × q2
      ⋈d(q2)(q1)
      q1 || q2
```

constant data
context value
unary operator
binary operator
map
select
cartesian product
dependent join
default



$$q @ d_1 \downarrow_a d_2 \leftrightarrow \sigma \vdash [q]_x \downarrow_c d_2$$

if $\sigma(x) = d_1$

Named Nested Relational Calculus

```
e ::= x
      d
      ⊕ e
      e1 ⊗ e2
      let x=e1 in e2
      {e2 | x ∈ e1}
      e1 ? e2 : e3
```

variables
constant data
unary operator
binary operator
let
comprehension
conditional

NNRC → CAMP

[[e]]

declarative



pattern matching

scrutinee

environment

x

renaming



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

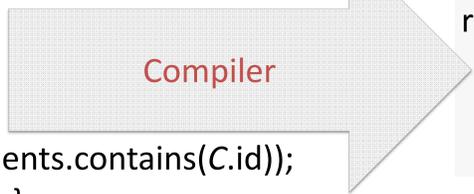


Everything in gray has been verified in Coq



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

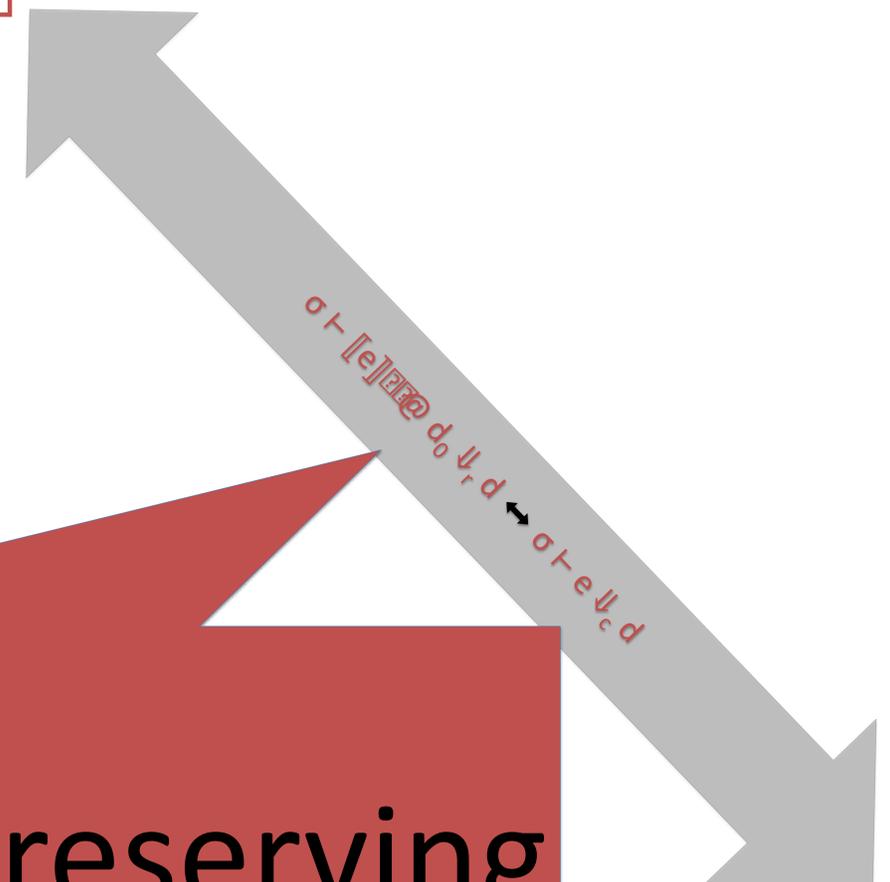
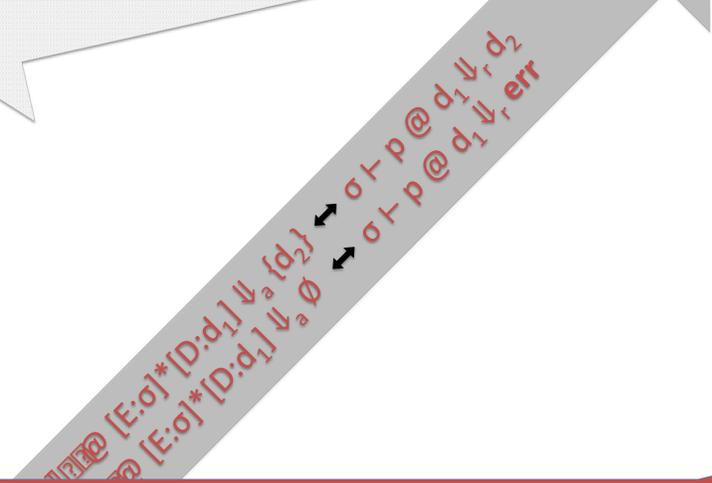
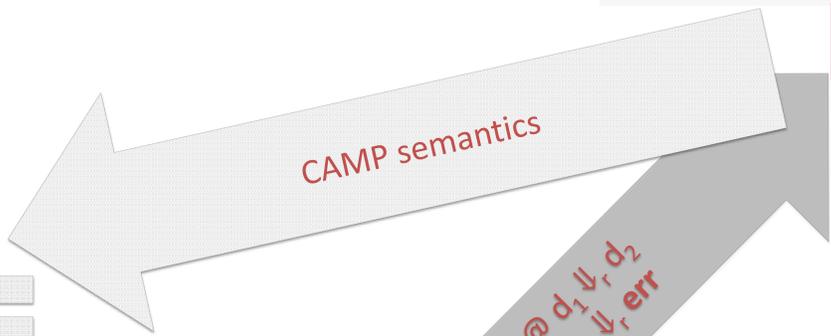
```
p ::= d
      | ⊕ p
      | p1 ⊗ p2
      | map p
      | assert p
      | p1 || p2
      | it | let it = p1 in p2
      | env | let env += p1 in p2
      | constant data
      | unary operator
      | binary operator
      | map over a bag
      | assertion
      | error recovery (orElse)
      | get/set scrutinee
      | 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

```
d1 ⊗ d2 ::=
  | d1 = d2 equality
  | d1 ∈ d2 element of
  | d1 ∪ d2 union
  | d1 * d2 biased record concat
  | d1 + d2 compatible record concat
```



Compilation is Semantics Preserving

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

$\sigma \vdash e \Downarrow_c d$

	Named	Nested	Relational	Calculus
e ::= x				variables
d				constant data
⊕ e				unary operator
e ₁ ⊗ 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

| ∞^d(q₂)(q₁) dependent join
| q₁ || q₂ 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);
  }
}
```



Rules

```
r ::= when p; r
      global p; r
      not p; r
      return p
```

Calculus for Aggregating Matching Patterns

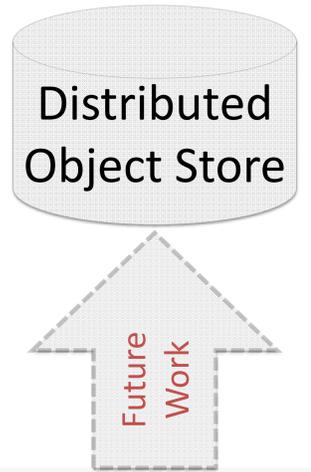
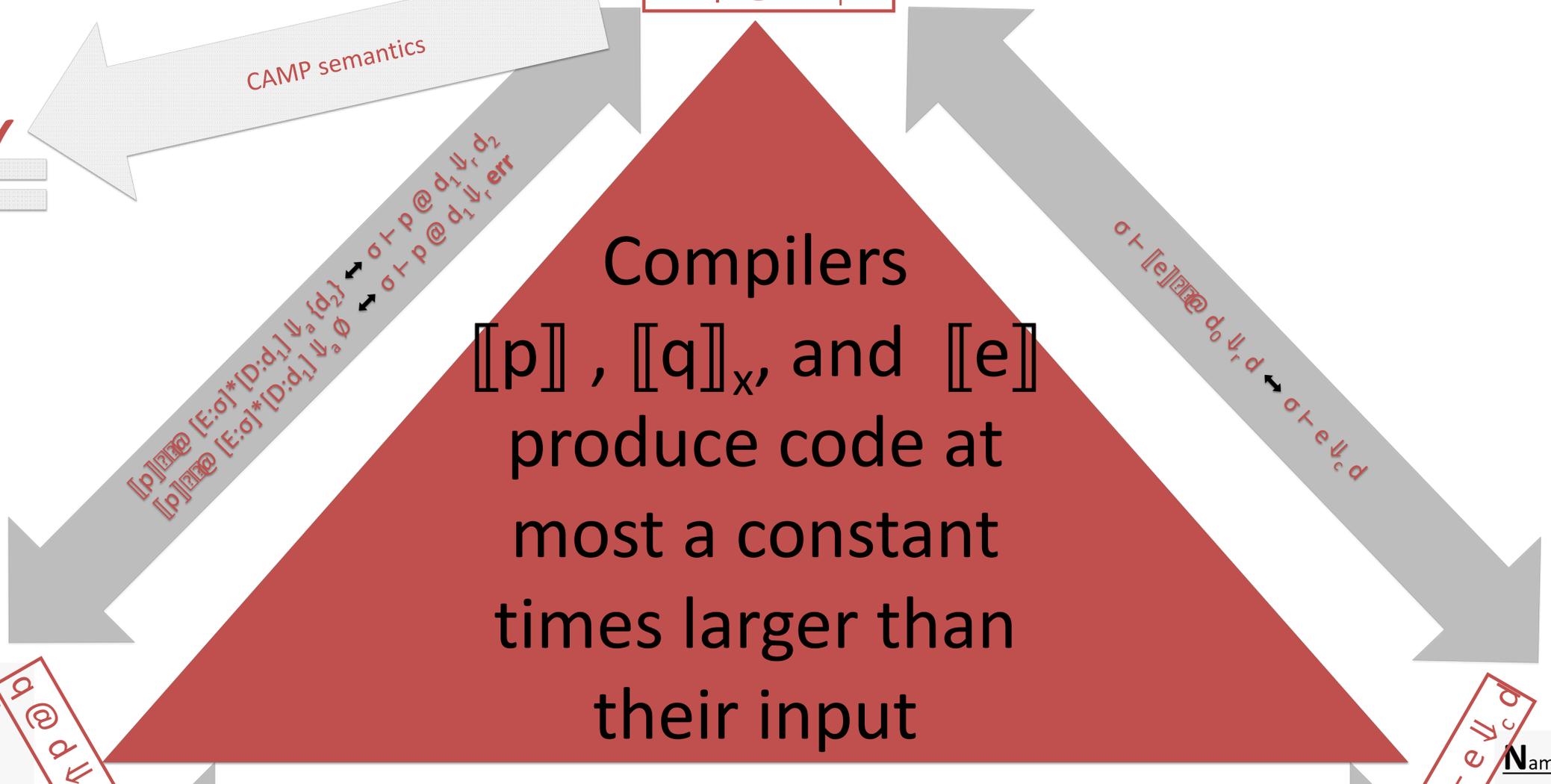
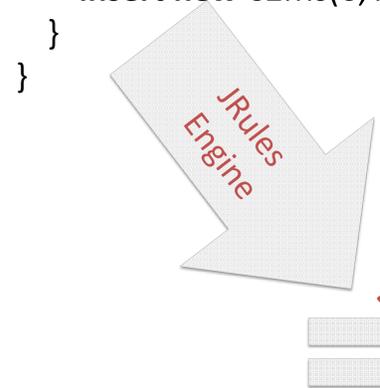
```
p ::= d
      ⊕ p
      p1 ⊗ p2
      map p
      assert p
      p1 || p2
      it | let it = p1 in p2
      env | let env += p1 in p2
      constant data
      unary operator
      binary operator
      map over a bag
      assertion
      error recovery (orElse)
      get/set scrutinee
      get/update 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

```
d1 ⊗ d2 ::=
  | d1 = d2 equality
  | d1 ∈ d2 element of
  | d1 ∪ d2 union
  | d1 * d2 biased record concat
  | d1 + d2 compatible record concat
```



Nested Relational Algebra

```
q ::= d
      In
      ⊕ q
      q1 ⊗ q2
      χ(q2)(q1)
      σ(q2)(q1)
      q1 × q2
      ⋈d(q2)(q1)
      q1 || q2
      constant data
      context value
      unary operator
      binary operator
      map
      select
      cartesian product
      dependent join
      default
```



$$q @ d_1 \downarrow_a d_2 \leftrightarrow \sigma \vdash [q]_x \downarrow_c d_2$$

Named Nested Relational Calculus

```
e ::= x
      d
      ⊕ e
      e1 ⊗ e2
      let x=e1 in e2
      {e2 | x ∈ e1}
      e1 ? e2 : e3
      variables
      constant data
      unary operator
      binary operator
      let
      comprehension
      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.contain
    } do { collect {M}; }
  } then {
    insert new C2Ms(C, Ms);
  }
}
```

Rules

Calculus for Aggregating Matching Patterns

Unary Operators

Binary Operators

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

- $\oplus d ::=$
- | identity d no-op. returns d
- | $\neg d$ 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

- $d_1 \otimes d_2 ::=$
- | $d_1 = d_2$ 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

$$\oplus d \Downarrow_o d$$

$$d \otimes d \Downarrow_o d$$

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

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

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

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

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

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
- | $\chi(q_2)(q_1)$ 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
- | **let** $x=e_1$ **in** e_2 let
- | $\{e_2 \mid x \in e_1\}$ comprehension
- | $e_1 ? e_2 : e_3$ conditional

$$\text{if } \sigma(x) = d_1 \text{ then } q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [q]_x \Downarrow_c d_2$$

$$q @ d \Downarrow_a d$$

$$\sigma \vdash e$$



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 $\exists d_1?$,

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

$$d_1? :_d \tau_1$$

Pattern Calculus for Aggregating Matching Patterns

- d constant data
- $\oplus d$ unary operator
- $\otimes d_1 d_2$ binary operator
- $\chi(q_2)(q_1)$ map over a bag
- $\sigma(q_2)(q_1)$ assertion
- $q_1 \times q_2$ error recovery (orElse)
- $\text{let } x = p_1 \text{ in } p_2$ get/set scrutinee
- $\text{let env} += p_1 \text{ in } p_2$ get/update environment

Unary Operators

- $\oplus d ::=$
- | identity d no-op. returns d
- | $\neg d$ 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

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

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

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

$$\oplus d \Downarrow_o d$$

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

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

$$d \otimes d \Downarrow_o d$$

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

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

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

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

Nested Relational Algebra

- $q ::= d$ constant data
- | **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

$$q @ d \Downarrow_a d$$

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

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

$$\text{if } \sigma(x) = d_1 \text{ then } q @ d_1 \Downarrow_a d_2 \leftrightarrow \sigma \vdash [q]_x \Downarrow_c d_2$$

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);
  }
}
```



Rules

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

Calculus for Aggregating Matching Patterns

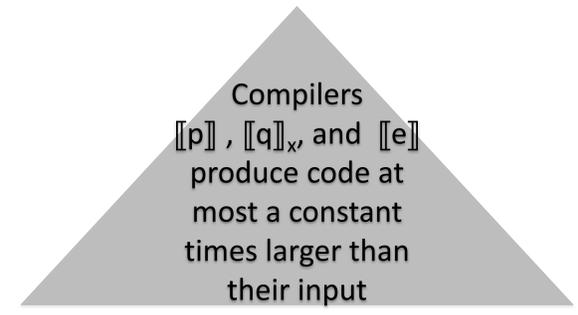
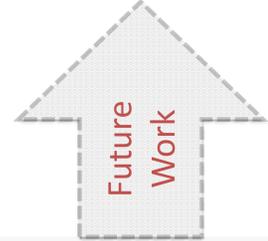
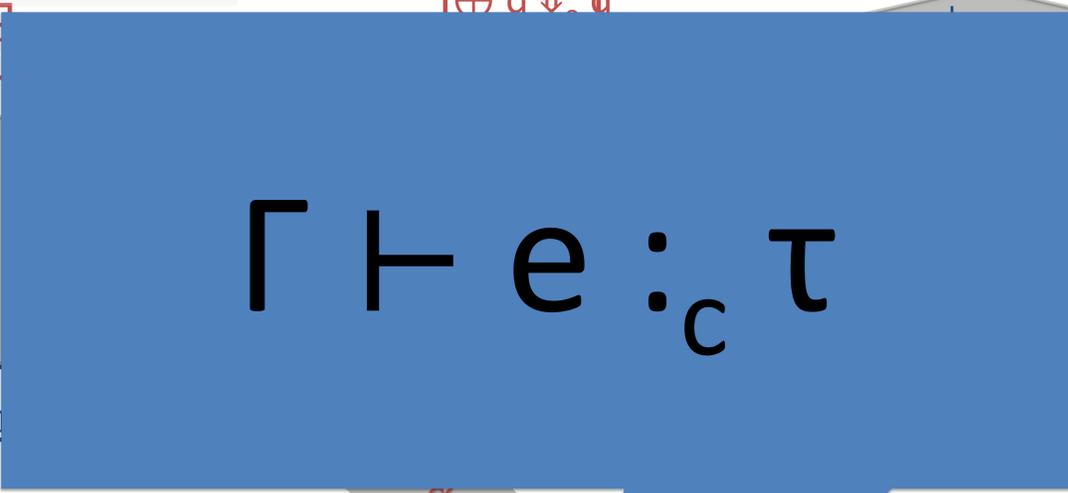
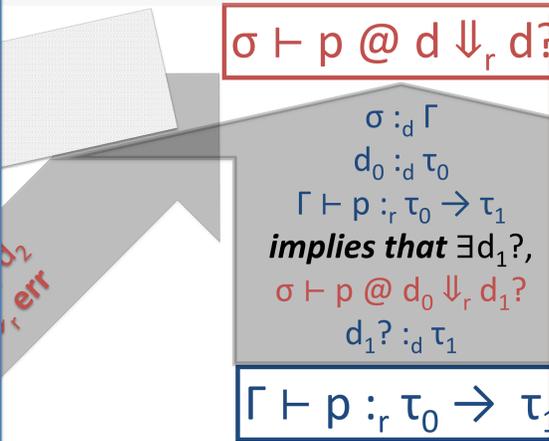
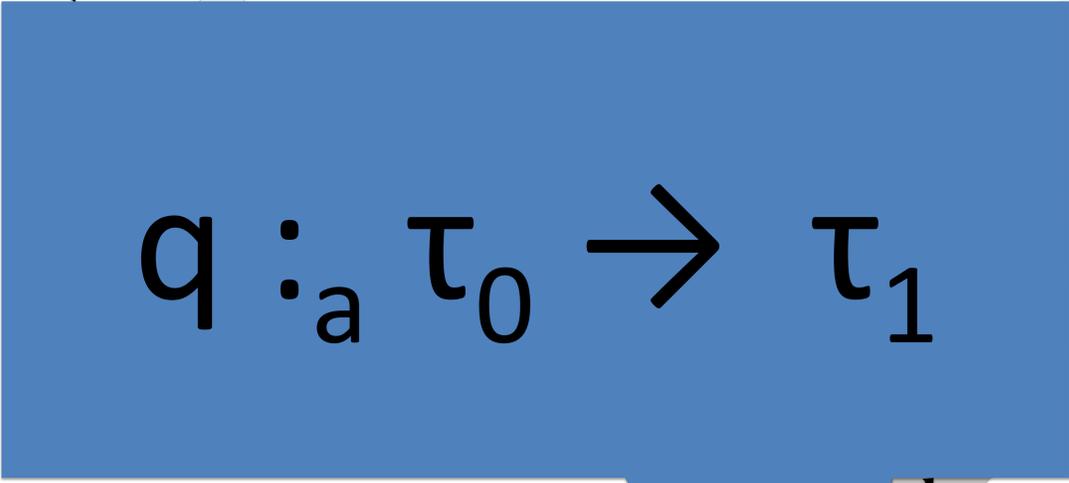
$p ::=$ d
 | $\oplus p$ constant data
 | $\text{unary } p$ unary operator
 | $p_1 \otimes p_2$ binary operator
 | map p map over a bag
 | assert p assertion
 | $p_1 || p_2$ error recovery (orElse)
 | it | let it = p_1 in p_2 get/set scrutinee
 | env | let env += p_1 in p_2 get/update environment

Unary Operators

$\oplus d ::=$
 | identity d no-op. returns d
 | $\neg d$ 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

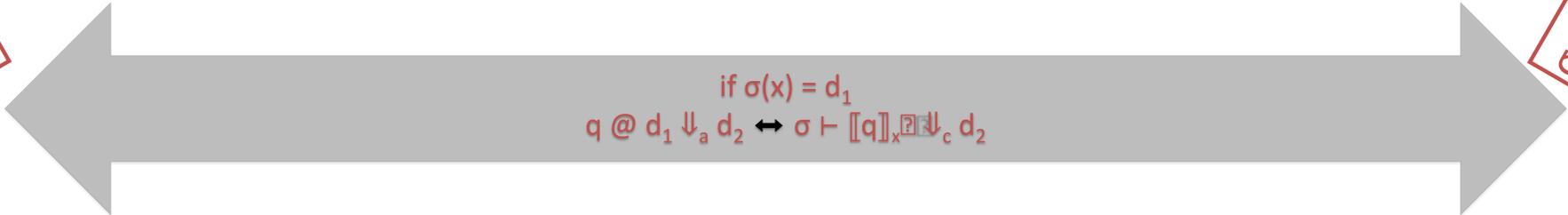
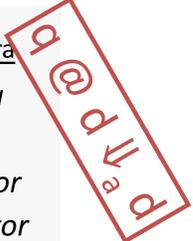
Binary Operators

$d_1 \otimes d_2 ::=$
 | $d_1 = d_2$ 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



Nested Relational Algebra

$q ::=$ d constant data
 | 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
 | let $x=e_1$ in e_2 let
 | $\{e_2 \mid x \in e_1\}$ 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 $\exists d,$

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

$$d :_d \tau$$

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

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

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

Distributed Object Store

Future Work

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

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

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

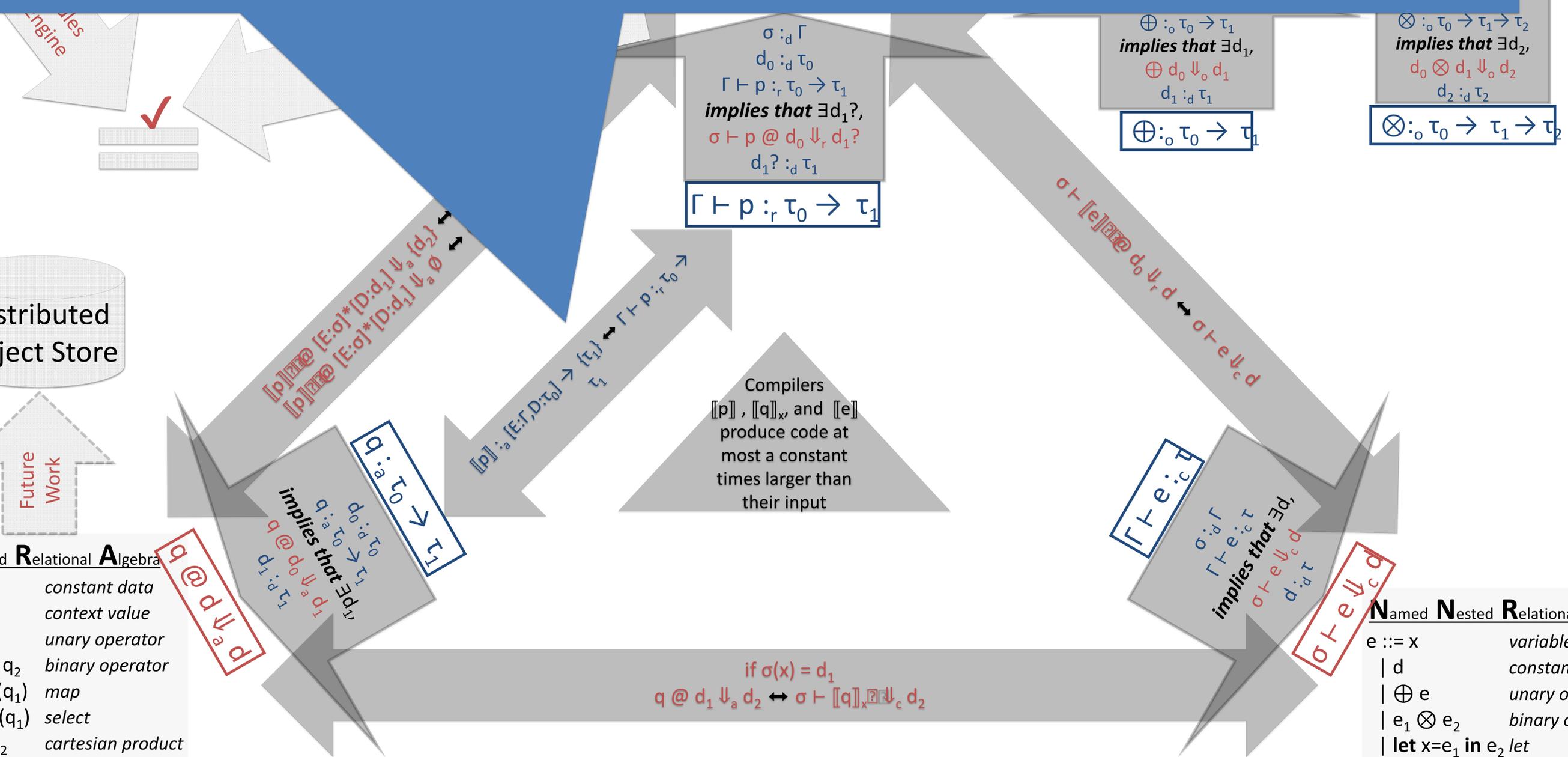


Type Preservation

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

rule Fir
when
C:
Ms
}
} the
ins
}

ors
rd concat
record concat
}



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

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$	let
$ \{e_2 \mid x \in e_1\}$	comprehension
$ e_1? e_2 : e_3$	conditional



Type Preservation

If $\Gamma(x) = \tau_0$

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

rule
wh
C
/
}
}tl
}
}

ators

of

cord concat
le record concat

o d

$\rightarrow \tau_2$



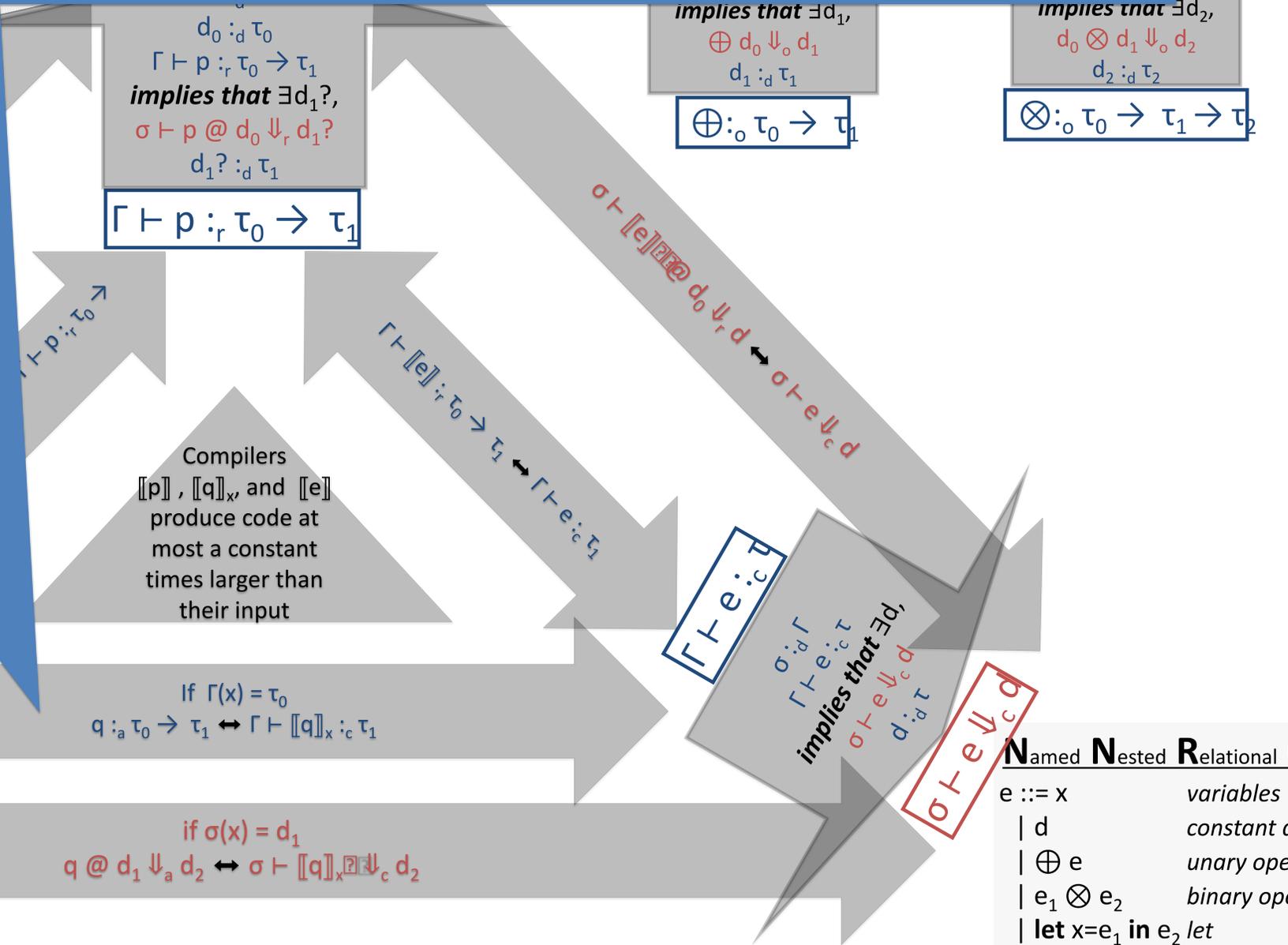
Future Work

Nested Relational Algebra

$q ::= d$	constant data
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
let $x=e_1$ in e_2	let
$\{e_2 \mid x \in e_1\}$	comprehension
$e_1 ? e_2 : e_3$	conditional





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.

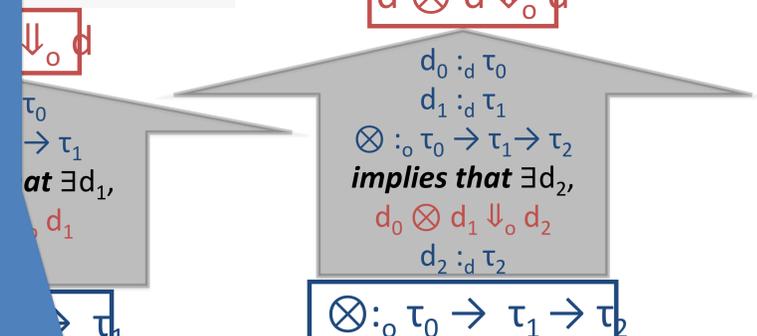
[1] J. Van den Bussche and S. Vansummen. Polymorphic type inference for the named nested relational calculus. *Transactions on Computational Logic (TOCL)*, 9(1), 2007.

Operators

p . returns d
 returns a Boolean
 returns a bag of d
 returns a bag of bags
 returns a constructor
 returns a selection
 returns a removal

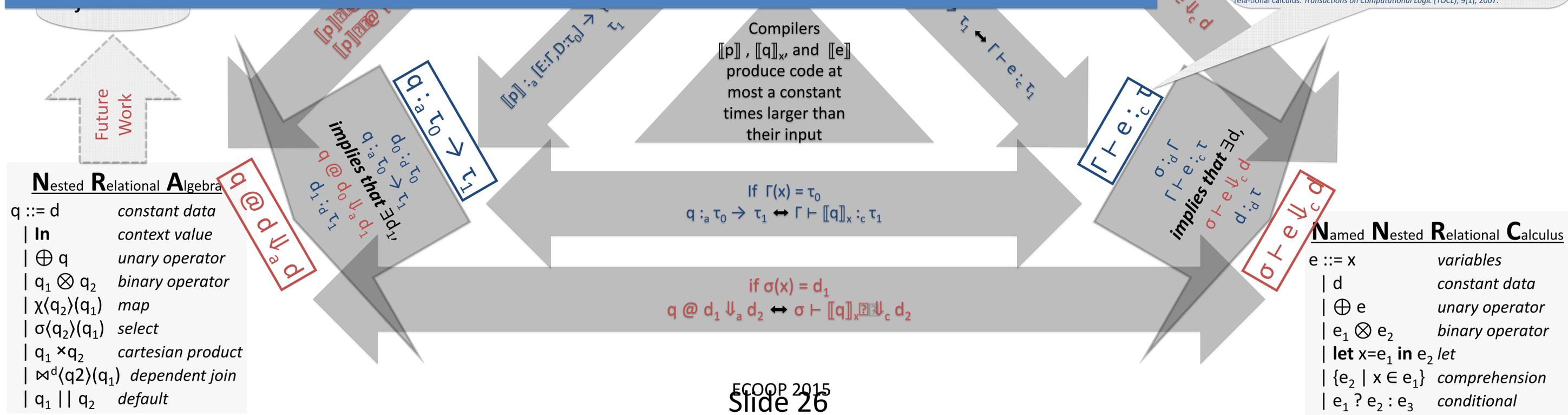
Binary Operators

$d_1 \otimes d_2 ::=$
 | $d_1 = d_2$ 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. Vansummen. Polymorphic type inference for the named nested relational 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
$\chi(q_2)(q_1)$	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

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

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

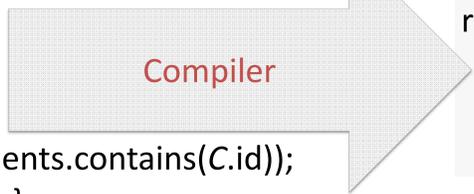


Everything in gray has been verified in Coq



IBM JRules

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

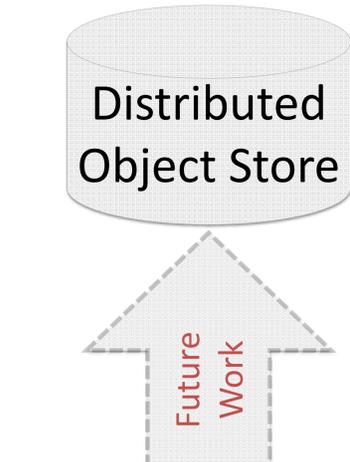
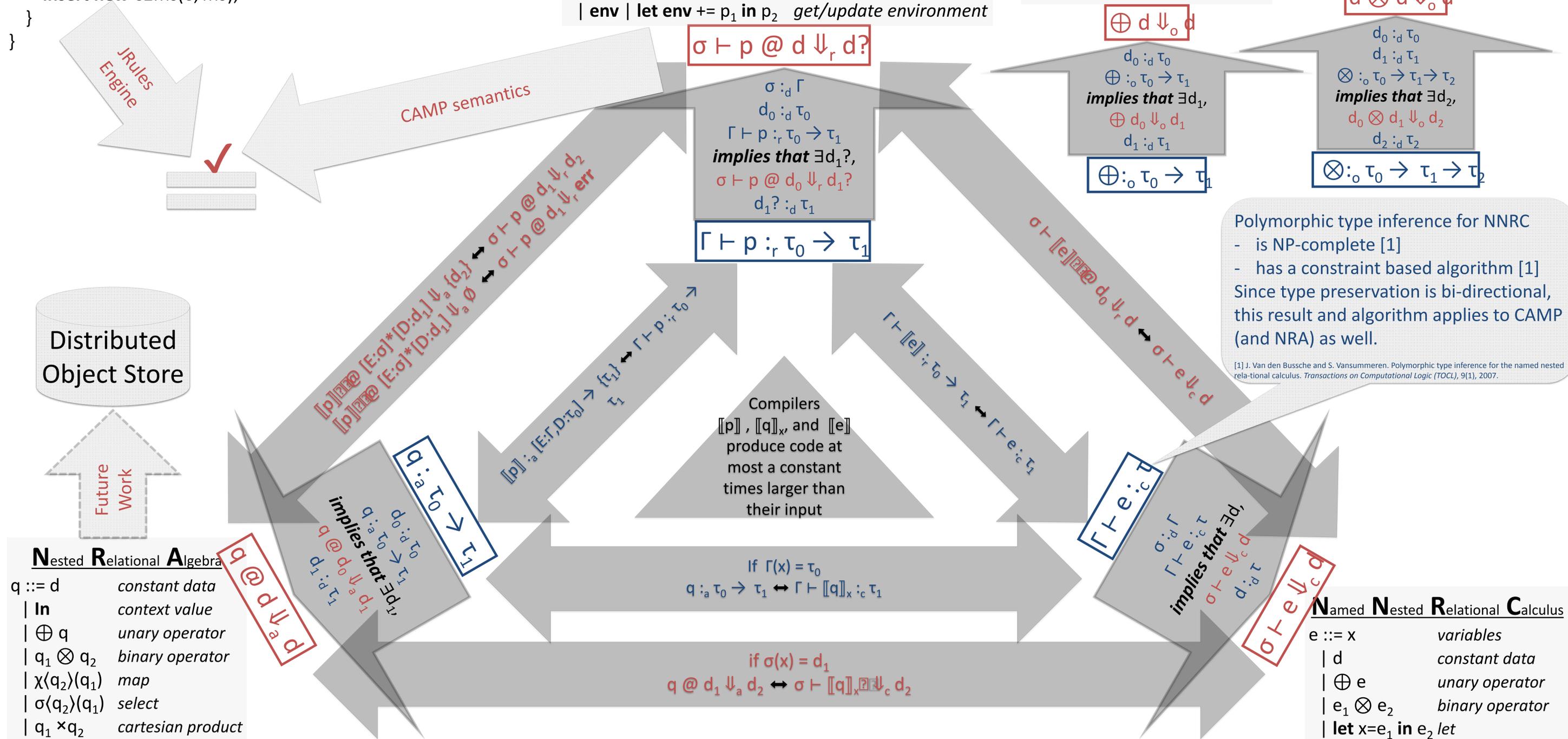


Rules
 $r ::= \text{when } p; r$
 $\text{global } p; r$
 $\text{not } p; r$
 $\text{return } p$

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

Unary Operators
 $\oplus d ::=$
 $\text{identity } d$ *no-op. returns d*
 $-d$ *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*



Nested Relational Algebra
 $q ::= d$ *constant data*
 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*
 $\text{let } x=e_1 \text{ in } e_2$ *let*
 $\{e_2 \mid x \in e_1\}$ *comprehension*
 $e_1 ? e_2 : e_3$ *conditional*