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

Avraham Shinnar, Jérôme Siméon, and Martin Hirzel

IBM Research

ECOOP ’15
**JRules example**

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

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

```java
class C2MS {
    Client C;
    List<Marketer> Ms;
}
```
Calculus for Aggregating Matching Patterns

\[ p ::= d \quad \text{constant data} \]
\[ \oplus p \quad \text{unary operator} \]
\[ p_1 \otimes p_2 \quad \text{binary operator} \]
\[ \text{map } p \quad \text{map over a bag} \]
\[ \text{assert } p \quad \text{assertion} \]
\[ p_1 \mid p_2 \quad \text{error recovery (orElse)} \]
\[ \text{it } | \text{let it } = p_1 \text{ in } p_2 \quad \text{get/set scrutinee} \]
\[ \text{env } | \text{let env } += p_1 \text{ in } p_2 \quad \text{get/update environment} \]
rule FindMarketers {
  when {
    C: Client();
    Ms: aggregate {
      M: Marketer(clients.contains(C.id));
    } do { collect {M}; }
  }
  then {
    insert new C2Ms(C, Ms);
  }
}
Rules

\[ r ::= \text{when } p; r \quad \text{Evaluate } p \text{ against each WME} \]
\[ \mid \text{global } p; r \quad \text{Evaluate } p \text{ once} \]
\[ \mid \text{not } p; r \quad \text{Ensure } p \text{ does hold for any WME} \]
\[ \mid \text{return } p \quad \text{Compute a result using } p \]
**IBM JRules**

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

**Compiler**

**CAMP semantics**

**IBM Research**

**A Pattern Calculus for Rule Languages:**
Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel

**Rules**

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

**Unary Operators**

\[ d ::= \text{identity} \ d \mid \neg d \mid \{d\} \mid \#d \mid \text{flatten} \ d \mid [A:d] \mid d.A \mid d^{-A} \mid d \oplus d \mid d \otimes d \]

**Binary Operators**

\[ d \otimes d ::= \text{equality} \mid d_1 \in d_2 \mid d_1 \cup d_2 \mid d_1 \times d_2 \mid d_1 \oplus d_2 \mid d_1 \otimes d_2 \]

**Unary Calculus for Aggregating Matching Patterns**

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

**Compiler**

\[ \sigma \vdash p @ d \downarrow, d? \]
**Nested Relational Algebra**

\[ q ::= d \quad \text{constant data} \]

\[ \text{In} \quad \text{context value} \]

\[ \ominus q \quad \text{unary operator} \]

\[ q_1 \otimes q_2 \quad \text{binary operator} \]

\[ \chi\langle q_2 \rangle(q_1) \quad \text{map} \]

\[ \sigma\langle q_2 \rangle(q_1) \quad \text{select} \]

\[ q_1 \times q_2 \quad \text{cartesian product} \]

\[ \bowtie^d \langle q2 \rangle(q_1) \quad \text{dependent join} \]

\[ q_1 \mid\mid q_2 \quad \text{default} \]
A Pattern Calculus for Rule Languages:
Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

IBM Jrules

Rules

| r ::= when p; r |
| global p; r |
| return p |

Compiler

CAMP semantics

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
| ω²(q₂)(q₁) dependent join
| q₁ || q₂ default

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 ::= equality
| d₁ ⊆ d₂ element of |
| d₁ ⊔ d₂ union |
| d₁ ∨ d₂ biased record concat |
| d₁ · d₂ compatible record concat |

Calculation for Aggregating Matching Patterns

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

σ ⊢ p @ d ⊪ r, d?
A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

**IBM JRules**

**Rules**

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

**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
| ⊕d | record constructor
| d.A | field selection
| d−A | field removal
```

**Binary Operators**

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

**Compiler**

```
σ ⊢ p @ d ⊥ r, d?
```

**Future Work**

```
Distributed Object Store
```

Slide 8
CAMP $\Rightarrow$ NRA

- Pattern matching
- Operational

- Scrutinee
- Environment that unifies repeated bindings
- Input

- \( d? = d + err \)
- Recoverable errors

- \( d \)
- \( \{d\} \)
- \( \emptyset \)
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: Marker(clients.contains(C.id)); 
    } do { collect (M); } 
  } then { 
    insert new C2Ms(C, Ms); 
  } 
}

Compiler

CAMP semantics

Distributed Object Store

Future Work

Nested Relational Algebra

<table>
<thead>
<tr>
<th>q ::=</th>
<th>constant data</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>unary operator</td>
</tr>
<tr>
<td>q</td>
<td>binary operator</td>
</tr>
<tr>
<td>q1 × q2</td>
<td>cartesian product</td>
</tr>
<tr>
<td>q1 ⊙ q2</td>
<td>dependent join</td>
</tr>
<tr>
<td>q1 ⊙ q2</td>
<td>default</td>
</tr>
</tbody>
</table>

Unary Operators

<table>
<thead>
<tr>
<th>d ::=</th>
<th>no-op. returns d</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬d</td>
<td>negates a Boolean</td>
</tr>
<tr>
<td>{d}</td>
<td>singleton bag of d</td>
</tr>
<tr>
<td>#d</td>
<td>size of bag</td>
</tr>
<tr>
<td>flatten d</td>
<td>flatten a bag of bags</td>
</tr>
<tr>
<td>[A:d]</td>
<td>record constructor</td>
</tr>
<tr>
<td>d.A</td>
<td>field selection</td>
</tr>
<tr>
<td>d−A</td>
<td>field removal</td>
</tr>
</tbody>
</table>

Binary Operators

<table>
<thead>
<tr>
<th>d ⊗ d</th>
<th>biased record concat</th>
</tr>
</thead>
<tbody>
<tr>
<td>d ⊗ d</td>
<td>compatible record concat</td>
</tr>
</tbody>
</table>

Compilation is Semantics Preserving

\[
\begin{align*}
\sigma \vdash p @ d \Downarrow a \{ d_2 \} & \iff \sigma \vdash p @ d_1 \Downarrow r d_2 \\
\sigma \vdash p @ E: \sigma, D: d_1 \Downarrow a \emptyset & \iff \sigma \vdash p @ d_1 \Downarrow r \text{err}
\end{align*}
\]
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

Slide 11
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
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);
  }
}

Calculus for Aggregating Matching Patterns

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

Unary Operators

⊕ d ::= |
| identity d
| ¬d
| {d}
| #d
| flatten d
| [A:d]

Binary Operators

d₁ ⊗ d₂ ::= |
| d₁ = d₂
| d₁ ∈ d₂
| d₁ ∪ d₂
| d₁ + d₂
| d₁ ⊥ d₂

Future Work

Distributed Object Store

Compiler

CAMP semantics

Verified

ECOOP 2015

Distinguished
Artifact
Award

Everything in gray has been verified in Coq
NRA ➔ NNRC

\[ [q]_x \]

operational \quad \Rightarrow \quad \text{declarative}

input \quad \Rightarrow \quad x \quad \text{environment}
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

Distributed Object Store

Compilation is Semantics Preserving

if σ(x) = d₁
q @ d₁ ↓ₐ d₂ ↔ σ ⊢ [q]ₓ ↓ₑ d₂

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

Unary Operators
� d::= d
| identity d
| ¬d
| {d}
| #d
| flatten d
| map p
| assert p
| p₁ | p₂
| it
| let it = p₁ in p₂
| env
| let env += p₁ in p₂

Binary Operators
� d ⊗ d::= d
| d₁ ⊗ d₂::= equality
| d₁ ⊠ d₂::= element of
| d₁ ⊥ d₂::= union
| d₁ + d₂::= biased record concat
| d ⊤ d::= compatible record concat

Future Work

ECOOP 2015

Slide 15
A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

IBM JRules

Rules

Calculate for Aggregating Matching Patterns

Unary Operators

Binary Operators

Distributed Object Store

Future Work

ECOOP 2015

Slide 16
NNRC ➔ CAMP

declarative ➔ pattern matching

environment ➔ scrutinee

x ➔ Environment that unifies repeated bindings

renaming ➔ recoverable errors

⟦e⟧ ➔ env.x
A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

**Unary Operators**

\[ e ::= x \quad \text{variables} \]
\[ \oplus e \quad \text{constant data} \]
\[ d \quad \text{no-op, returns } d \]
\[ \neg d \quad \text{negates a Boolean} \]
\[ \{d\} \quad \text{singleton bag of } d \]
\[ #d \quad \text{size of bag} \]
\[ \text{flatten } d \quad \text{flatten a bag of bags} \]
\[ [A:d] \quad \text{record constructor} \]
\[ d.A \quad \text{field selection} \]
\[ d-A \quad \text{field removal} \]

**Binary Operators**

\[ \oplus d \quad \text{bias record concat} \]
\[ d \oplus d \quad \text{compatible record concat} \]

---

**Compilation is Semantics Preserving**

\[ \sigma \vdash \sem{e} \downarrow_r d \iff \sigma \vdash e \downarrow_c d \]
Compilers \([p]\), \([q]\), and \([e]\) produce code at most a constant times larger than their input.
Type Soundness

\[ \sigma : d \Gamma \]
\[ d_0 : d \tau_0 \]
\[ \Gamma \vdash p : r \tau_0 \rightarrow \tau_1 \]

implies that \( \exists d_1 ?, \sigma \vdash p \at d_0 \downarrow_r d_1 ? \)
\[ d_1 ? : d \tau_1 \]
Type Soundness

\[
\begin{align*}
\sigma : d &
\Gamma \vdash e : c \tau \\
\text{implies that } &
\exists d_1, \\
\sigma @ d_1 &
\downarrow_c d
\end{align*}
\]

\[
\begin{align*}
d_1 : d &
\tau_1
\end{align*}
\]
Type Preservation

\[
[p] : A [E : \Gamma, D : \tau_0] \rightarrow \{\tau_1\} \iff \Gamma \vdash p : \tau_0 \rightarrow \tau_1
\]
Type Preservation

If $\Gamma(x) = \tau_0$

$$q : a \tau_0 \rightarrow \tau_1 \iff \Gamma \vdash \llbracket q \rrbracket_x : c \tau_1$$
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.

A Pattern Calculus for Rule Languages: Expressiveness, Compilation, and Mechanization
Avraham Shinnar, Jérôme Siméon, and Martin Hirzel
IBM Research

Rules

Compiler

Distributed Object Store

IBM JRules

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

Calculi for Aggregating Matching Patterns

Unary Operators

Binary Operators

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.


ECOOP 2015