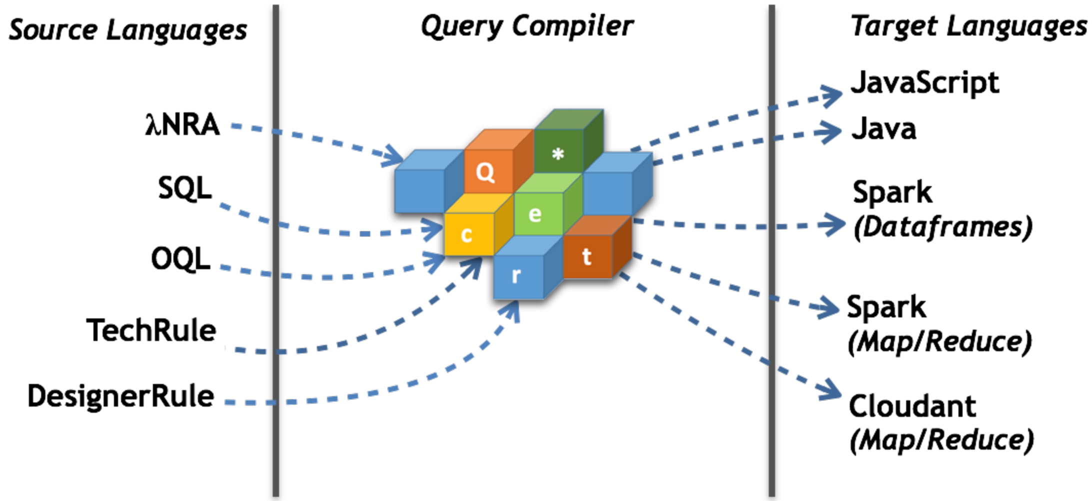
Handling Environments in a Nested Relational Algebra with Combinators and an Implementation in a Verified Query Compiler

Joshua Auerbach, Martin Hirzel, Louis Mandel, Avi Shinnar and Jérôme Siméon

IBM T.J. Watson Research Center

ACM SIGMOD, May 2017

1. Verified Query Compiler



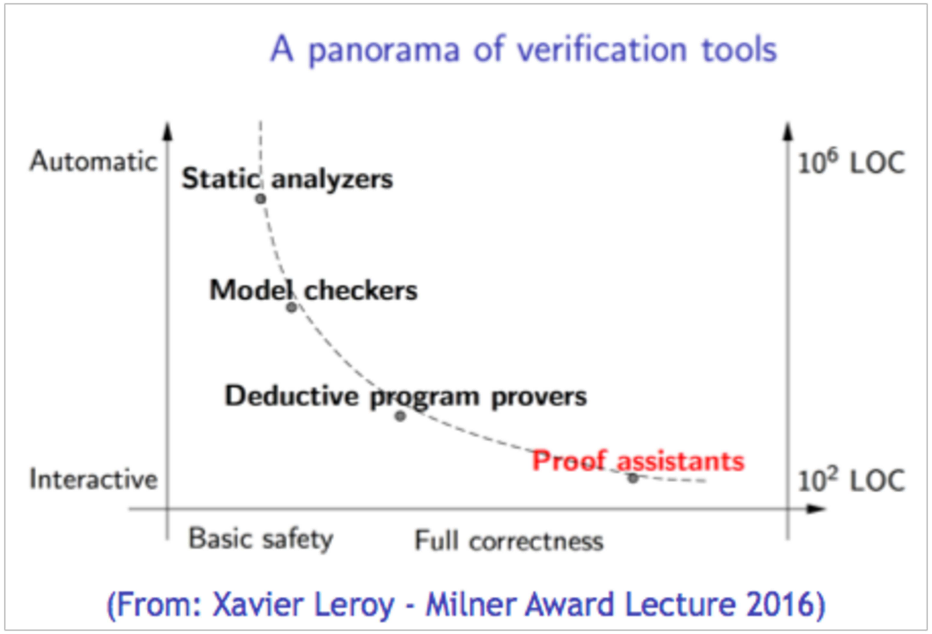
Why?

- Bugs are costly, security & privacy, guarantee access control, ...
- Define and check new optimizations
- Specify and compile new languages (e.g., DSLs)

How?

- Implemented with the Coq Proof Assistant
- Proof that the compiler preserves semantic is machine-checked

1. Verified Query Compiler



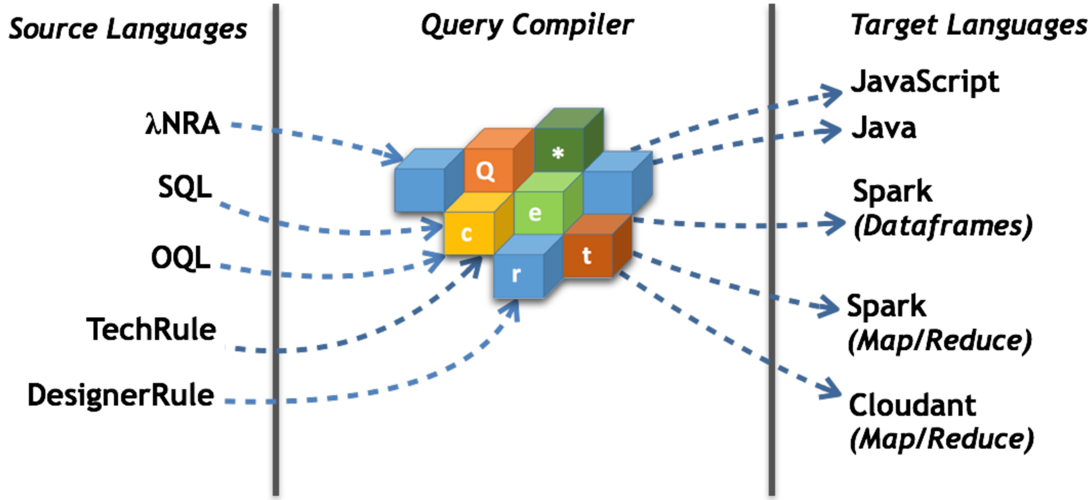
Some recent successes:

- CompCert (C compiler) ; Yxv6 (file system)
- seL4 (secure microkernel) ; HACMS program (secure drones)

Database-related: (Also: Cosette at SIGMOD'2017)

- DataCert [ESOP'2014]; mini-XQuery [CPP'2011]; RDBMS [POPL'2010]
- Optimizer Generator in Coko-Kola Project [SIGMOD 1996,1998...]

1. Verified Query Compiler



SQL:

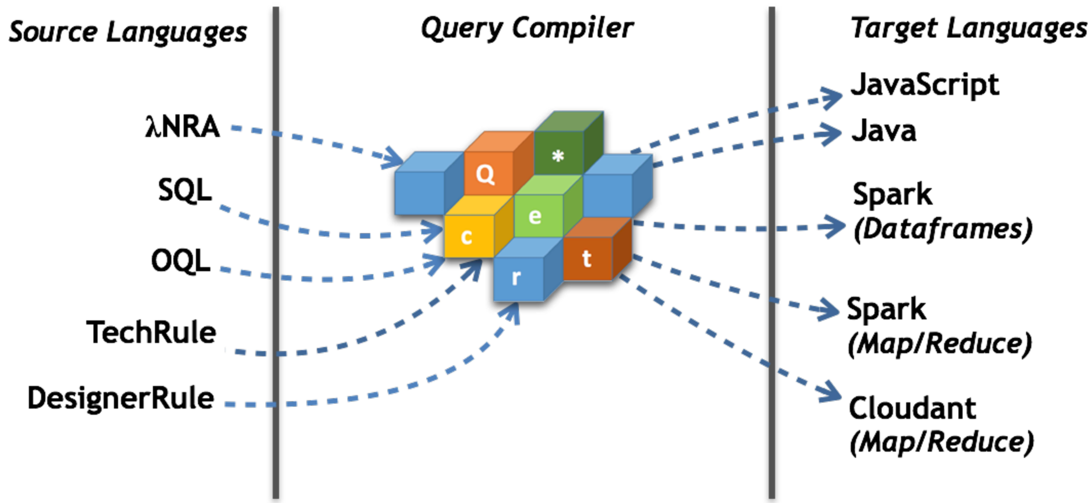
```
SELECT *
FROM (SELECT name,age FROM employees
       UNION ALL
      SELECT name,age FROM students) AS persons
WHERE age = 32;
```

OQL (yes ODMG one):

```
define view as
  select struct(name:c->name, purchased: p->name)
  from c in Clients, p in Purchases
  where p->cid = c->id;

select x.name from x in view where x.purchased = "Tomatoe"
```

1. Verified Query Compiler



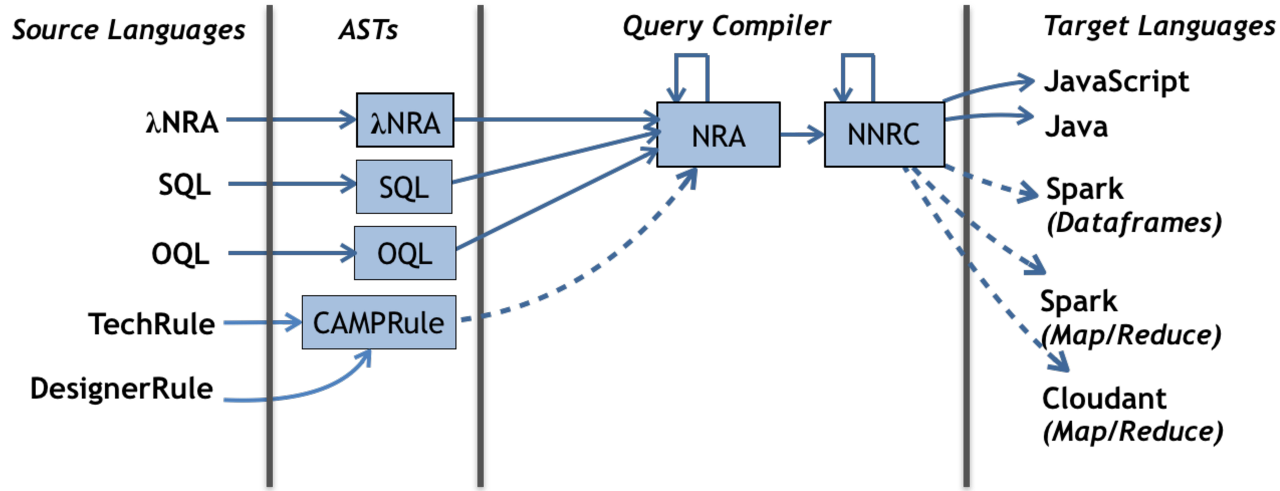
NRA^{λ}:

```
Customers.filter{ p => p.age = 32 }.map{ p => p.salary }.avg()
```

IBM's ODM Insights Designer Rules:

```
define 'test05' as detailed below, evaluated every minute.
definitions
  set 'test05' to the number of Customers,
       where the age of each Customer equals 32;
use 'test05' as the result.
```

1. Verified Query Compiler



- Like any other query compiler:
  - Source to AST to Logical Algebra to Physical Plan to Code
  - Emitted code executed by runtime (e.g., JVM, Database)
- But:
  - Each intermediate language needs a complete formal semantics
  - Logical: Nested Relational Algebra (from Cluet & Moerkotte)
  - "Physical": Named Nested Relational Calculus (from Van den Bussche & Vansummeren).

1. Verified Query Compiler

```
Lemma tselect_union_distr q₀ q₁ q₂ : (* Equivalence *)
  σ⟨ q₀ ⟩(q₁ ∪ q₂) ⇒ σ⟨ q₀ ⟩(q₁) ∪ σ⟨ q₀ ⟩(q₂).
Proof. ... Qed.

Definition select_union_distr_fun q := (* Functional rewrite *)
  match q with
  | NRAEnvSelect q0 (NRAEnvBinop AUnion q1 q2) =>
      NRAEnvBinop AUnion (NRAEnvSelect q0 q1) (NRAEnvSelect q0 q2)
  | _ => q
  end.

Proposition select_union_distr_fun_correctness q: (* Rewrite is correct *)
  select_union_distr_fun q ⇒ q.
Proof.
  tprove_correctness q.
  apply tselect_union_distr.
Qed.
```

Challenges:

- Depth of specification (equality, what's an equivalence, typing...).
- Handling environments in intermediate representations

2. Handling Environments

With variables (i.e., lambdas):

$$\textbf{map}(\lambda a.(a.city))(\textbf{map}(\lambda p.(p.addr))(P)) \equiv \textbf{map}(\lambda p.((p.addr).city))(P)$$

$$\textbf{map}(\lambda x.(e))(\textbf{map}(\lambda y.(u))(v)) \equiv \textbf{map}(\lambda y.(\underline{e[u/x]}))(v)$$

Without variables (i.e., combinators):

$$\chi_{\langle \textbf{In}.a.city\rangle}\big(\chi_{\langle[a:\textbf{In}]\rangle}\big(\chi_{\langle \textbf{In}.p.addr\rangle}\big(\chi_{\langle[p:\textbf{In}]\rangle}(q)\big)\big)\big) \equiv \chi_{\langle \textbf{In}.p.addr.city\rangle}\big(\chi_{\langle[p:\textbf{In}]\rangle}(q)\big)$$

$$\chi_{\langle q_1\rangle}\big(\chi_{\langle q_2\rangle}(q)\big) \equiv \chi_{\big\langle \underline{q_1 \circ q_2}\big\rangle}(q)$$

- Rewrites with variables/binders is harder (e.g., here involves substitution)
- Rewrites with combinators is easier (e.g., here plan composition)
- Correctness of binders manipulation notoriously difficult to mechanize
- See: POPLMark, and once again, Cherniack and Zdonik, SIGMOD 1996!

2. Handling Environments

With variables (i.e., lambdas):

$$\mathbf{map}(\lambda \underline{p}.([p : \underline{p},\ k : \mathbf{filter}(\lambda \underline{c}.(\underline{p}.age < \underline{c}.age))(\underline{p}.child)])(P)$$

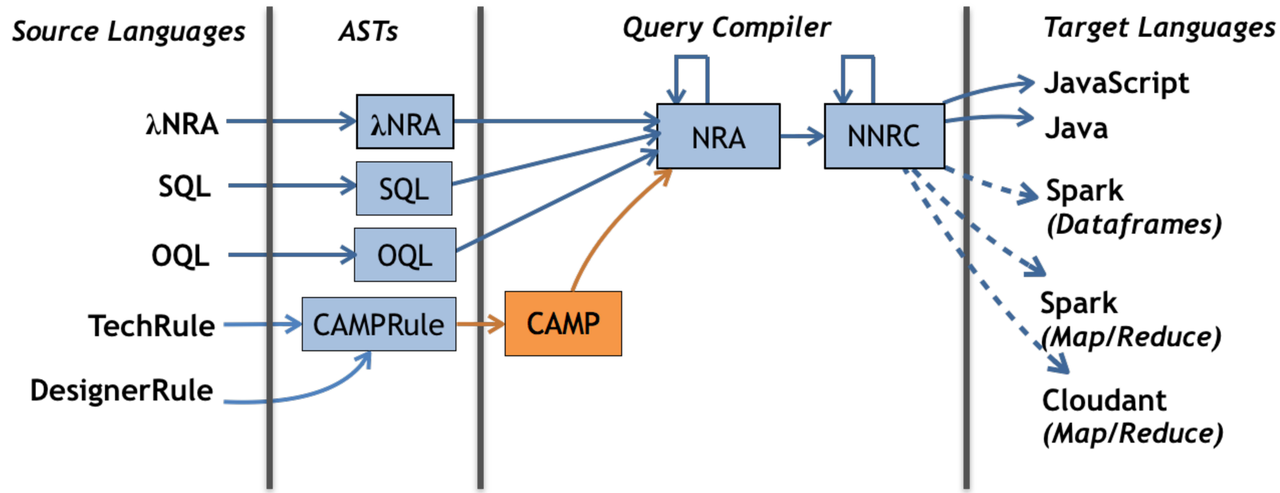
Without variables (i.e., combinators):

$$\chi_{\left\langle \left[p : \underline{\mathbf{In}.p},\ k : \chi_{\langle \underline{\mathbf{In}.c} \rangle}\left(\sigma_{\langle \underline{\mathbf{In}.p}.age < \underline{\mathbf{In}.c}.age \rangle}\left(\bowtie^{d}_{\left\langle \chi_{\langle \underline{[c:\mathbf{In}]} \rangle}(\underline{\mathbf{In}.p}.child) \right\rangle}(\{\mathbf{In}\}) \right) \right) \right] \right\rangle}\left(\chi_{\langle \underline{[p:\mathbf{In}]} \rangle}(P) \right)$$

Cost of reification:

- 5 iterators instead of 2
- nesting depth 3 instead of 2
- Use of dependent join ($\bowtie^{d}$) to combine p and c bindings

2. Handling Environments

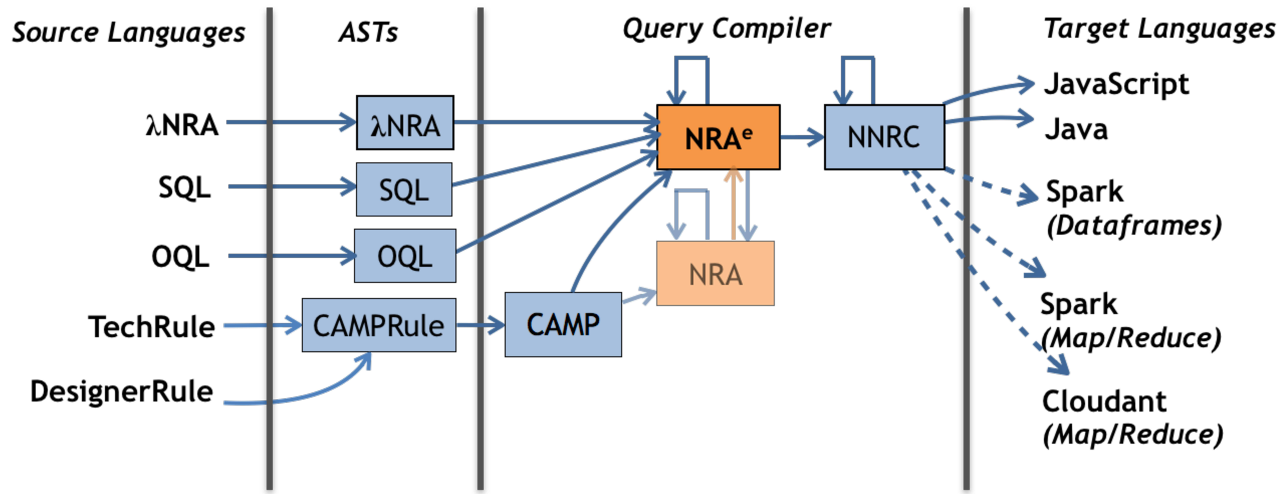


Sensitive to source language semantics & Encoding

e.g., for Designer Rules DSL:

- Environment = Source language variables + current item being matched
- Initial plans: from 400 to 2500 operators, depth 7 to 13
- Reification of environment manipulation impedes optimization

3. Nested Relational Algebra with Combinators

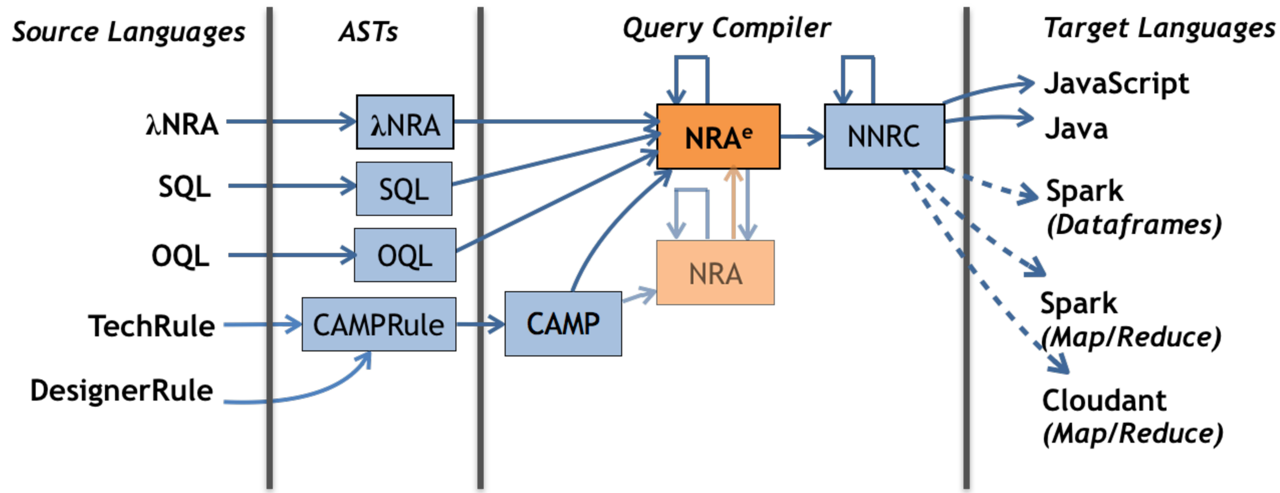


NRA Syntax $q \quad ::= \quad d \mid \mathbf{In} \mid q_2 \circ q_1 \mid \boxplus\, q \mid q_1 \boxtimes q_2 \mid \chi_{\langle q_2 \rangle}(q_1)$
$\mid \sigma_{\langle q_2 \rangle}(q_1) \mid q_1 \times q_2 \mid \bowtie^d{}_{\langle q_2 \rangle}(q_1) \mid q_1 \parallel q_2$

NRA Semantics $\vdash q \,@\, d \Downarrow_a d'$

- $\vdash$ **In** $@\ \boxed{\mathrm{d}} \Downarrow_a d$ (current value)
- $q_1 \circ q_2$ (sets current value in q_1 to q_2)
- $\boxplus\, q$: flatten, $q.a$, π, ...; $q_1 \boxtimes q_2$: $q_1 = q_2$, $q_1 \cup q_2$, $\oplus$ (record concatenation), ...
- χ (map) ; σ (selection) ; $\times$ (Cartesian product) ; $\bowtie^d$ (dependent join)

3. Nested Relational Algebra with Combinators



$$\textbf{NRA}^e\ \textbf{Syntax}\quad q \quad ::= \quad d \mid \textbf{In} \mid q_2 \circ q_1 \mid \boxplus\, q \mid q_1 \boxtimes q_2 \mid \chi_{\langle q_2 \rangle}(q_1)$$
$$\mid \sigma_{\langle q_2 \rangle}(q_1) \mid q_1 \times q_2 \mid \bowtie^d{}_{\langle q_2 \rangle}(q_1) \mid q_1 \parallel q_2$$
$$\mid \underline{\textbf{Env}} \mid \underline{q_2 \circ^e q_1} \mid \underline{\chi^e_{\langle q \rangle}}$$

$$\textbf{NRA}^e\ \textbf{Semantics}\quad \underline{\gamma} \vdash q \,@\, d \Downarrow_a d'$$

- $\boxed{\gamma} \vdash$ **Env** $@\, d \Downarrow_a \gamma$ (current environment)
- $q_1 \circ^e q_2$ (sets current environment in q_1 to q_2)
- $q_1 \circ^e$ **Env** $\oplus [x : q_2]$ (adding x to environment)

3. Nested Relational Algebra with Combinators

With variables (i.e., lambdas):

$$\textbf{map}(\lambda \underline{p}.([p : \underline{p},\ k : \textbf{filter}(\lambda \underline{c}.(\underline{p}.age < \underline{c}.age))(\underline{p}.child)])(P)$$

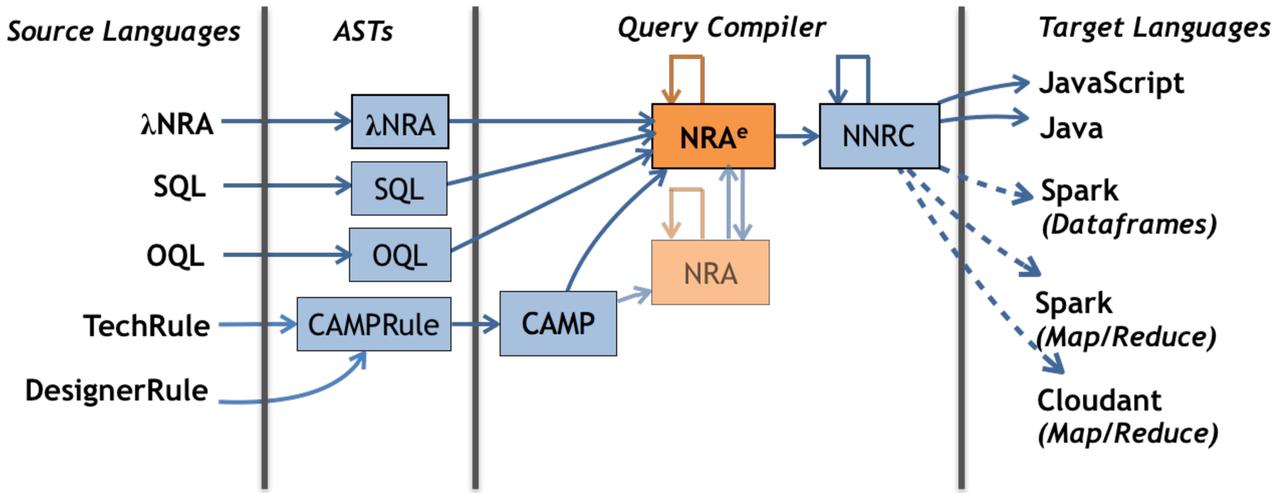
With NRA^e:

$$\chi_{\left\langle [p : \underline{\textbf{Env}.p},\ k : \sigma_{\left\langle (\underline{\textbf{Env}.p}.age < \underline{\textbf{Env}.c}.age) \circ^e (\underline{\textbf{Env} \oplus [c : \textbf{In}]}) \right\rangle}\left(\underline{\textbf{Env}.p}.child\right)] \circ^e \underline{[p : \textbf{In}]} \right\rangle}(P)$$

Cost of reification:

- Same number of iterators: 2
- Same nesting depth: 3
- No added (dependent) join

3. Nested Relational Algebra with Combinators



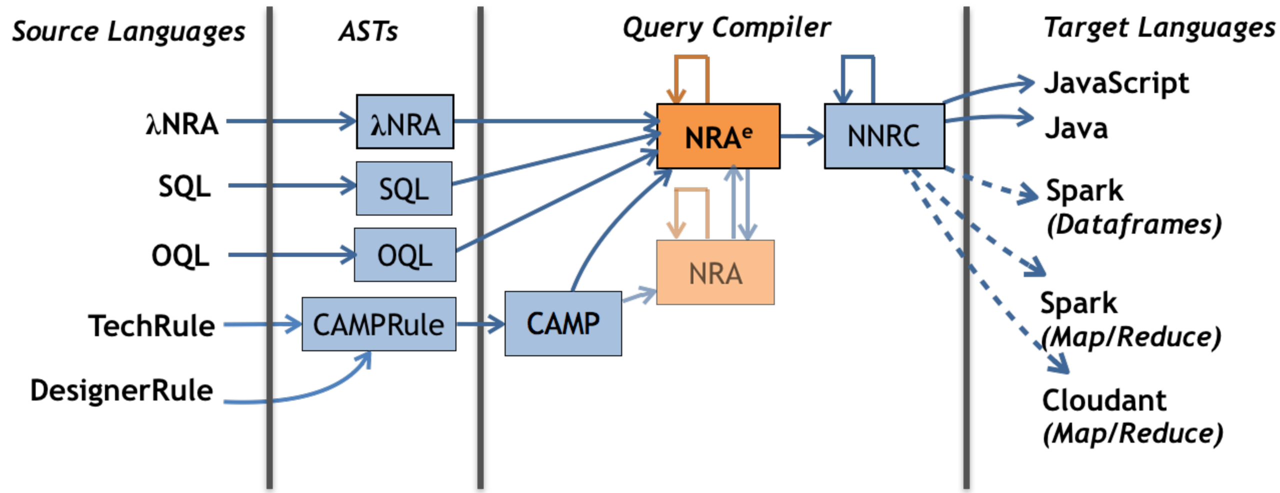
Lifting theorem

- All existing equivalences for NRA carry over to NRA^e

$$\sigma_{\langle q_0 \rangle}(q_1 \cup q_2) \equiv \sigma_{\langle q_0 \rangle}(q_1) \cup \sigma_{\langle q_0 \rangle}(q_2)$$

$$\chi_{\langle q_1 \rangle}\big(\chi_{\langle q_2 \rangle}(q)\big) \equiv \chi_{\langle q_1 \circ q_2 \rangle}(q)$$

3. Nested Relational Algebra with Combinators



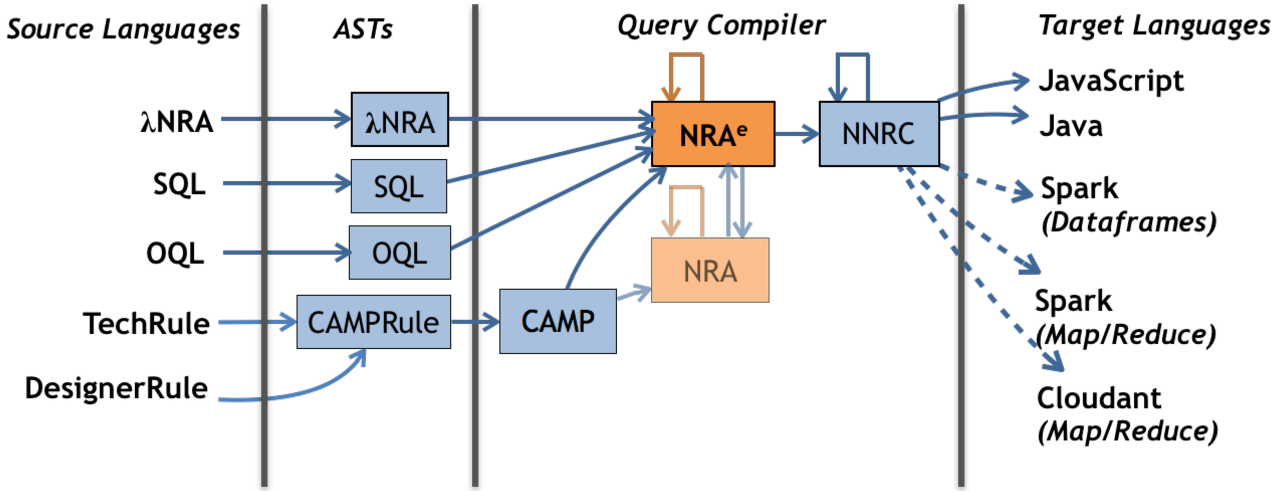
Lifting theorem

- All existing equivalences for NRA carry over to NRA^e
- True even is sub-plans parameters contain NRA^e operators!

$$\forall q_1, q_2, q \in \mathsf{NRA}, \chi_{\langle q_1 \rangle}\big(\chi_{\langle q_2 \rangle}(q)\big) \equiv_a \chi_{\langle q_1 \circ q_2 \rangle}(q)$$

$$\Longrightarrow \forall q_1, q_2, q \in \mathsf{NRA}^e, \chi_{\langle q_1 \rangle}\big(\chi_{\langle q_2 \rangle}(q)\big) \equiv_e \chi_{\langle q_1 \circ q_2 \rangle}(q)$$

3. Nested Relational Algebra with Combinators



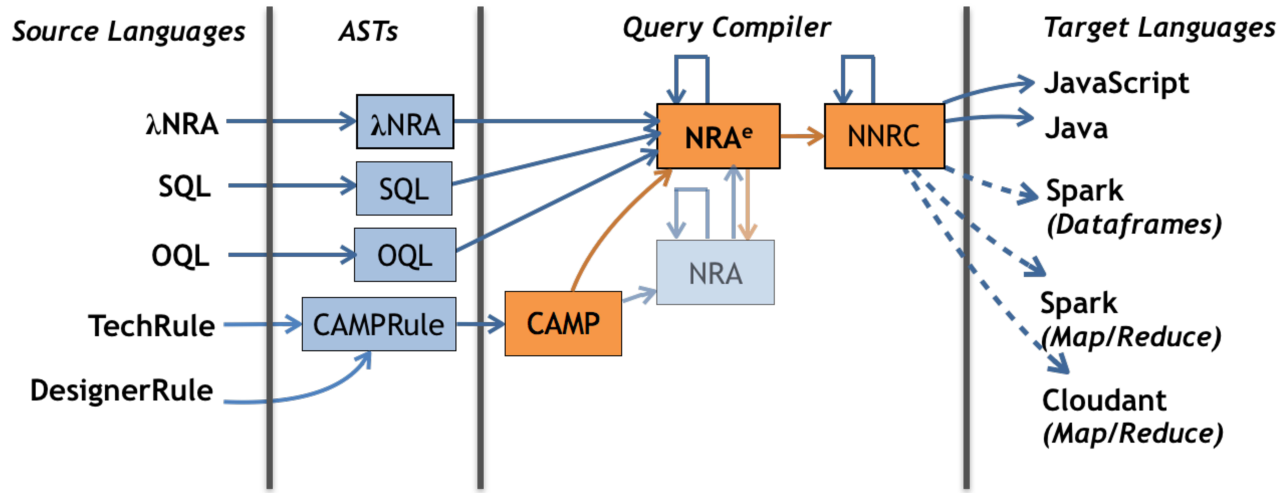
Lifting theorem

- Yes, the proof of that theorem has been mechanized

```
Fixpoint lift_nra_context (c:nra_ctxt) : nraenv_core_ctxt := ...

Theorem contextual_equivalence_lifting (c₁ c₂:nra_ctxt) :
  c₁ ≡ₐ c₂ -> lift_nra_context c₁ ≡ₑ lift_nra_context c₂.
Proof.
  apply lift_nra_context_proper.
Qed.
```

3. Nested Relational Algebra with Combinators



Translations In-Out of NRA^e

- from NRA^e to NNRC and NRA
- from CAMP and NRA^λ(without blowup) to NRA^e

3. Nested Relational Algebra with Combinators

- from NRA^e to NRA in LaTeX

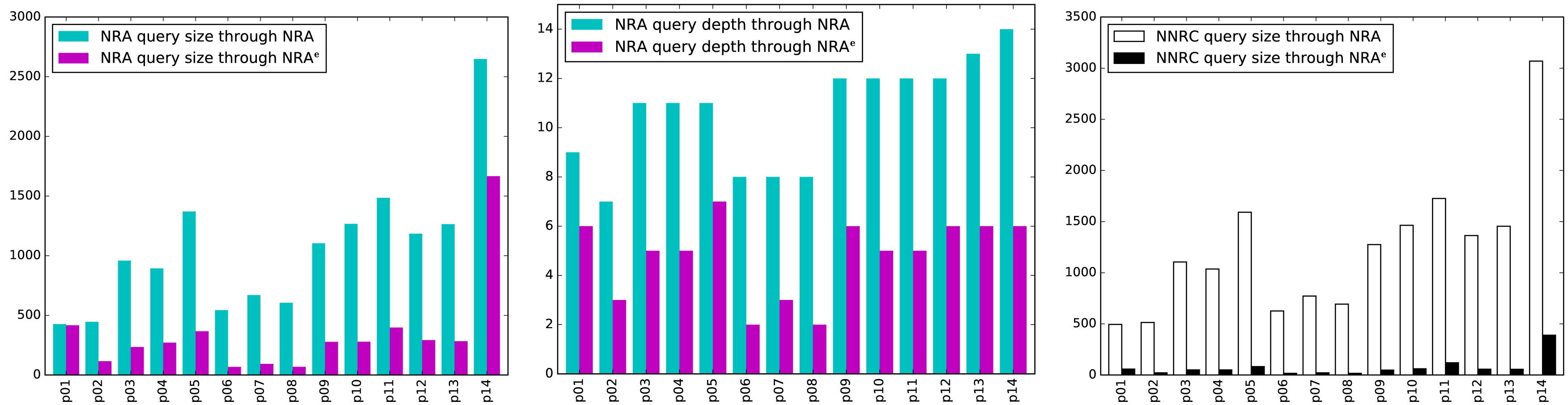
$$
\begin{aligned}
[\![d]\!]_a &= d \\
[\![\mathbf{In}]\!]_a &= \mathbf{In}.D \\
[\![q_2 \circ q_1]\!]_a &= [\![q_2]\!]_a \circ ([E : \mathbf{In}.E] \oplus [D : [\![q_1]\!]_a]) \\
[\![\boxplus q]\!]_a &= \boxplus [\![q]\!]_a \\
&\dots
\end{aligned}
$$

Figure 4: From NRA^e to NRA ✿. $\boxed{[\![q]\!]_a = q'}$

- from NRA^e to NRA in Coq (+ correctness proofs)

```
Fixpoint nra_of_nraenv_core (ae:nraenv_core) : nra :=
  match ae with
    | ANID => nra_data
    | ANConst d => (AConst d)
    | ANApp ea1 ea2 => AApp (nra_of_nraenv_core ea1)
                            (nra_wrap (nra_of_nraenv_core ea2))
    | ANUnop u ae1 => AUnop u (nra_of_nraenv_core ae1)
...
```

3. Nested Relational Algebra with Combinators

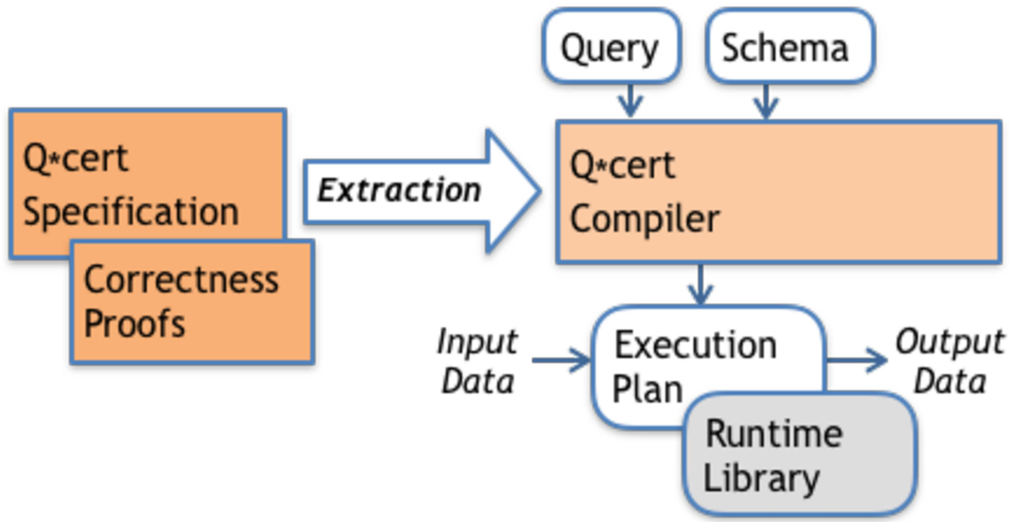


(a) Initial plan sizes. (b) Initial plan depths. (c) Emitted NNRC sizes.

Other Practical Benefits:

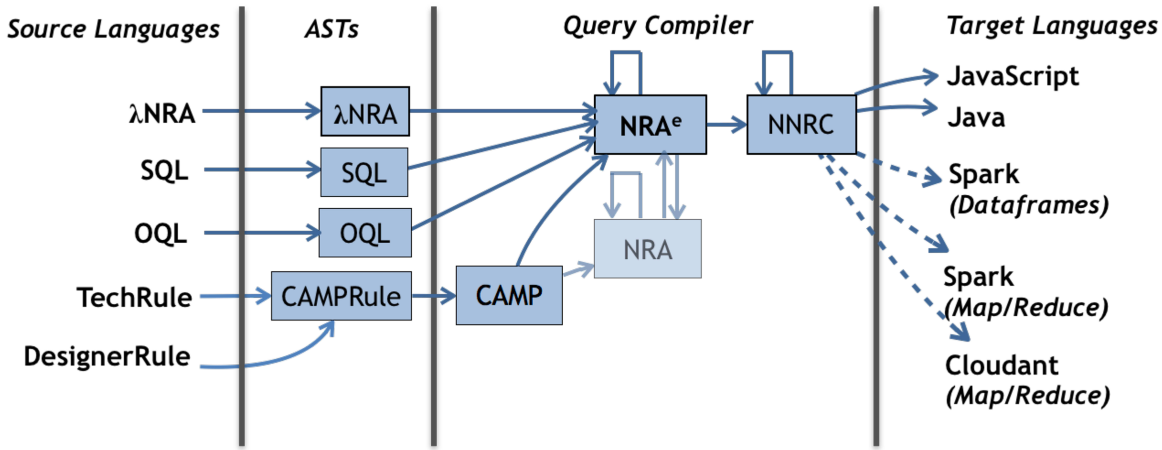
- NRA^e gives an elegant way to represent 'let' bindings:
  - e.g., view definitions for SQL and OQL ($q \circ^e$ **Env** $\oplus [view : q_v]$)
  - e.g., common subexpression elimination in query plans
- Optimization for ODM Designer Rules
  - Combination of existing and new NRA rewrites (~100)
  - Benchmarks: plan size and depth; Optimizer effectiveness

4. Implementation



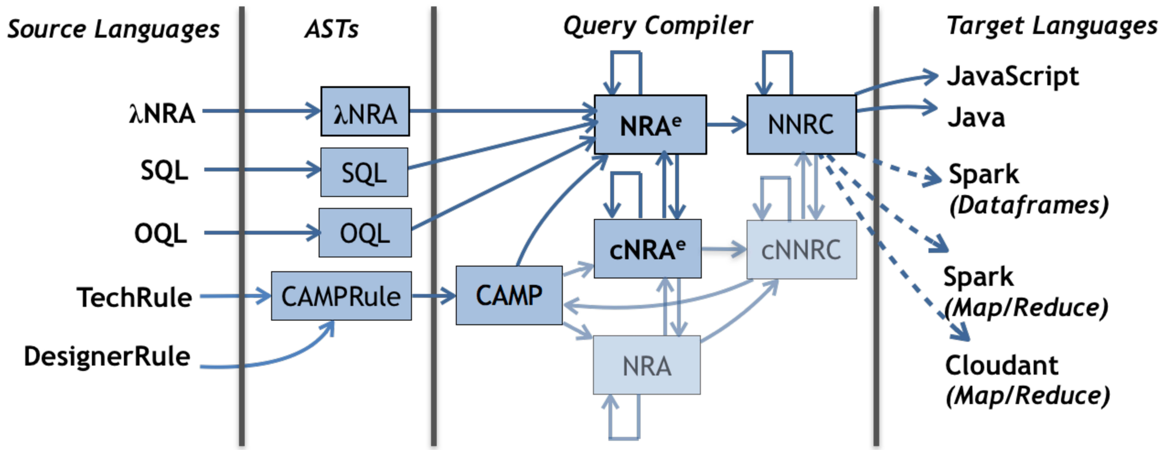
- Around: 40k lines of code ; 45k lines of proofs
- Coq $\mapsto$ OCaml (90k) $\mapsto$ native code or JavaScript
- Optimizer: naive cost model, directed rewrites, until fixpoint
  - Designed for extensibility (add/remove optimizations; change cost)
  - Timed up to a few seconds for large plans (e.g., TPC-H queries)
- Small runtimes for now (Java, Javascript and Scala)

4. Implementation



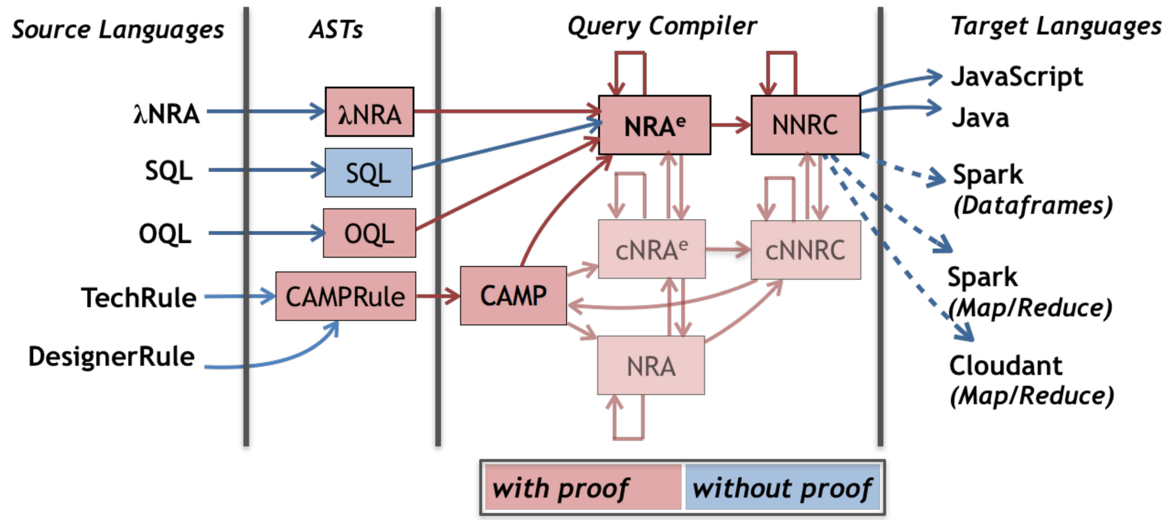
- Type System (Wadlerfest'2016):
  - Support for objects (used in OQL and Designer rules)
  - Support for optional types (e.g., for null values)

4. Implementation



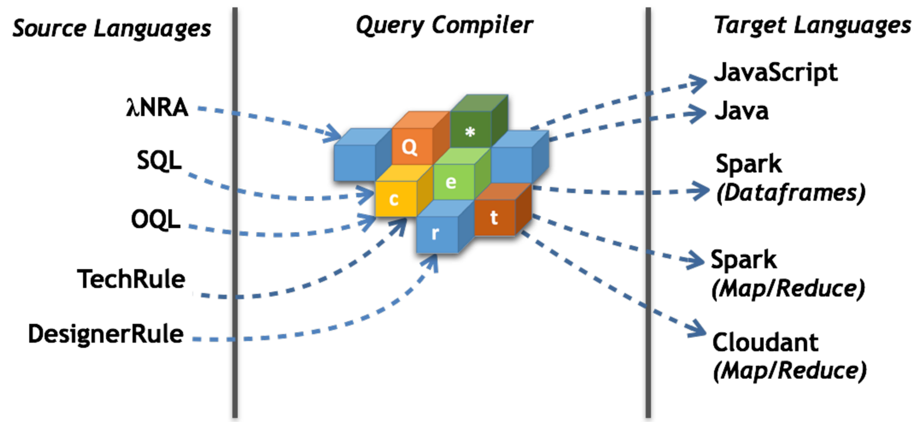
- Type System (Wadlerfest'2016):
  - Support for objects (used in OQL and Designer rules)
  - Support for optional types (e.g., for null values)
- Full NRA^e with OrderBy, GroupBy and Joins

4. Implementation



- Type System (Wadlerfest'2016):
  - Support for objects (used in OQL and Designer rules)
  - Support for optional types (e.g., for null values)
- Full NRA^e with GroupBy and Joins
- Proof coverage matters & garbage-in garbage-out

Conclusion



http://querycert.github.io/

- Query compiler in Coq; Large subset of compiler proved correct
- NRA^e: Easier rewrites & proofs ; Keep plan simple
- Some future directions: (suggestions or applications welcome!)
  - End to end certification (e.g., SQL to JavaScript)
  - Certified runtimes (including e.g., Join algorithms)
  - Other languages (e.g., SQL++) or backend
  - Grow the query optimizer (Join reordering, Cost model...)