Prototyper un compilateur de requêtes avec Coq

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

IBM T.J. Watson Research Center

January 2017

ODM Insights

IBM Operational Decision Manager - Decision Server Insights :

- interprète les événements provenant de clients
- réagit intelligemment et en temps réel

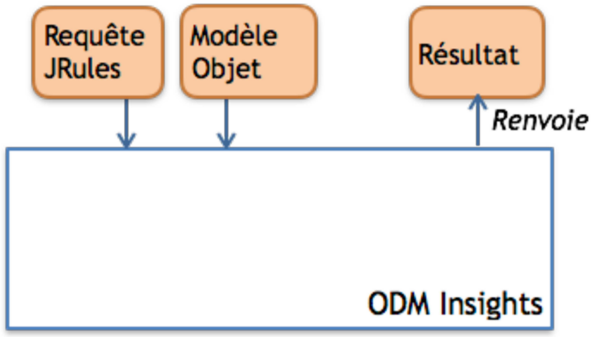
Types d'applications :

- détection de fraude
- campagnes marketing
- gestion d'alertes

Problématiques :

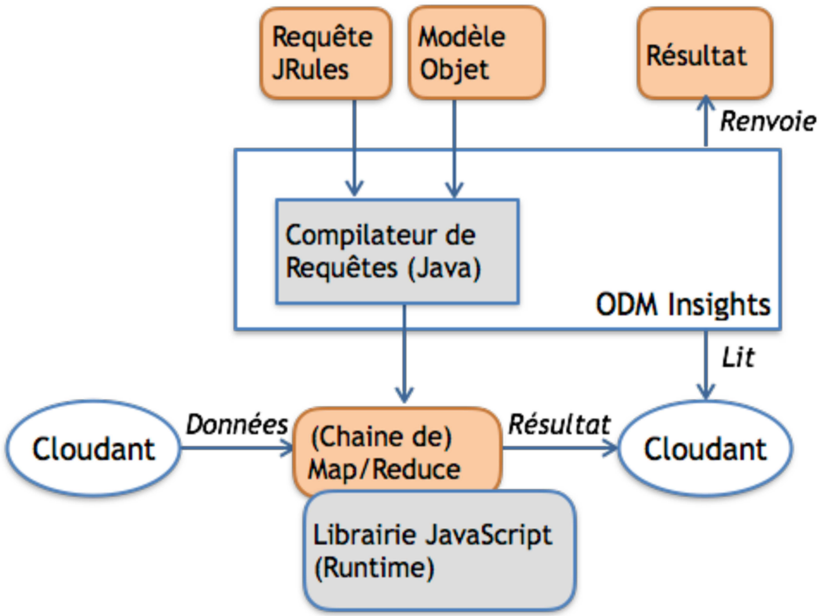
- langages JRules lisibles par des experts métier
- gestion d'évenements + requêtes
- passage à l'échelle

ODM Insights : Une requête




```
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.
```

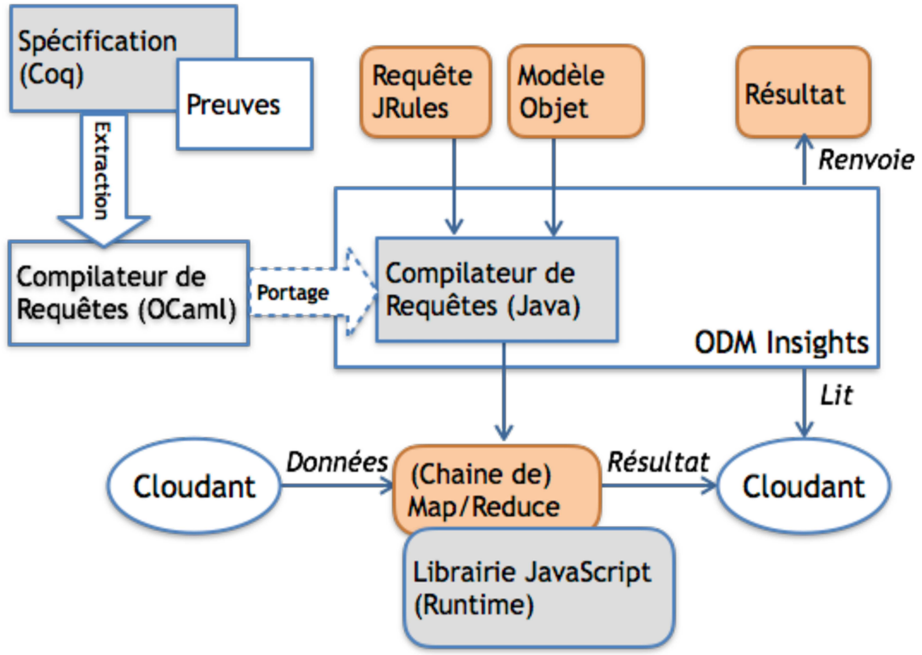
ODM Insights




```
{ "designs": [
    { "dbname": "test05b",
      "design": { "views": { "test05b": {
        "map": "function (doc) {\n  ... emit(0,srcout[iout]); ... }",
        "reduce": "_count"... } } } }
  ], ... }
```

Au final : Compilateur en Java, maintenu par l'équipe de dévelopement d'ODM

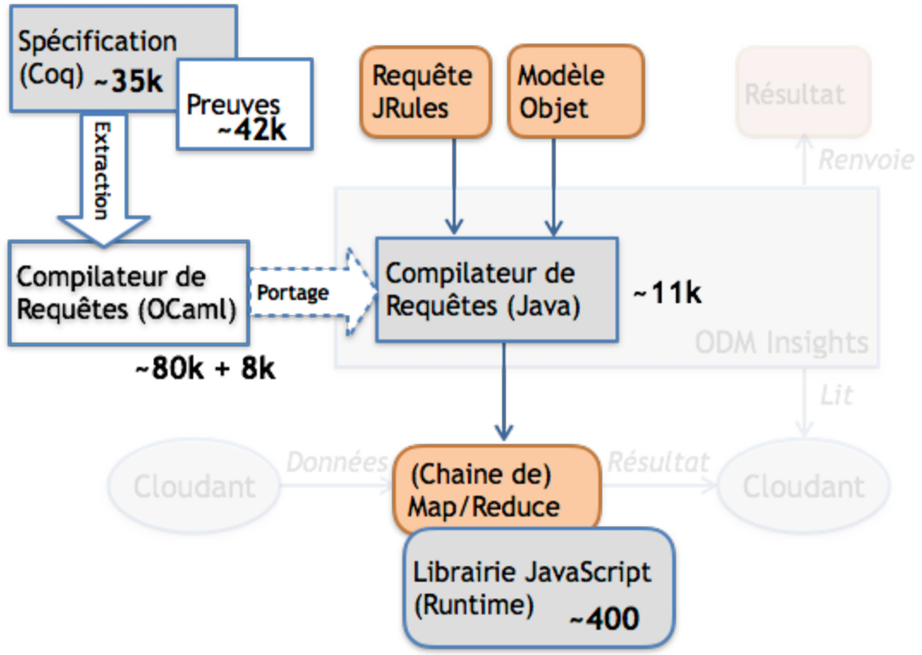
Prototypage en Coq



Pourquoi Coq ?

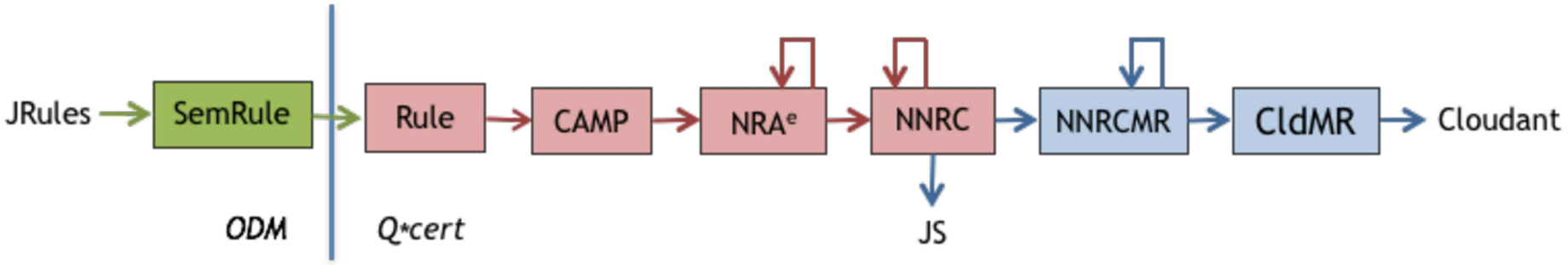
- Langage fonctionnel
- Distance sémantique importante entre langage source et cible
- Vérifier les optimisations (dont certaines inhabituelles)
- Intéret de l'équipe de recherche

Quelques Chiffres



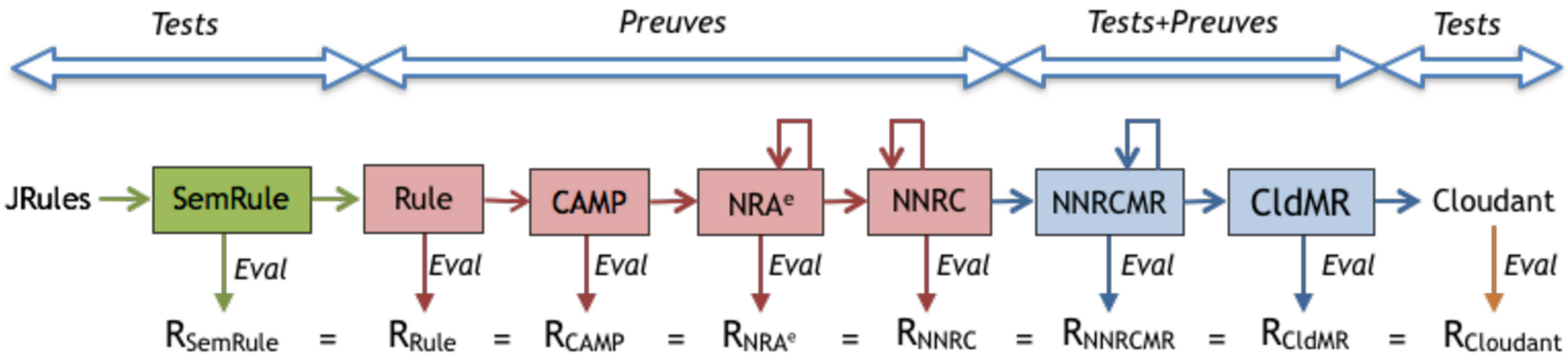
- 2014 : Sémantique de JRules et traduction vers une algèbre base de données (7k spec, 10k preuves)
- 2015 : Compilateur complet (optimiseur, introduction de map/reduce, génération de code)
- 2016 : Intégration dans ODM (couverture, portage vers Java, tests) + Compilateur Recherche open-source
- Total : environ 4 années-personnes

Le compilateur de requêtes



- De SemRule (filtrage par motifs, model à objets)
- Vers Cloudant (Base de données distribuée pour JSON, map/reduce)
- En passant par l'algèbre relationelle imbriquée (NRA^e) pour l'optimisation
- Six langages intermédiaires, trois passes d'optimisation

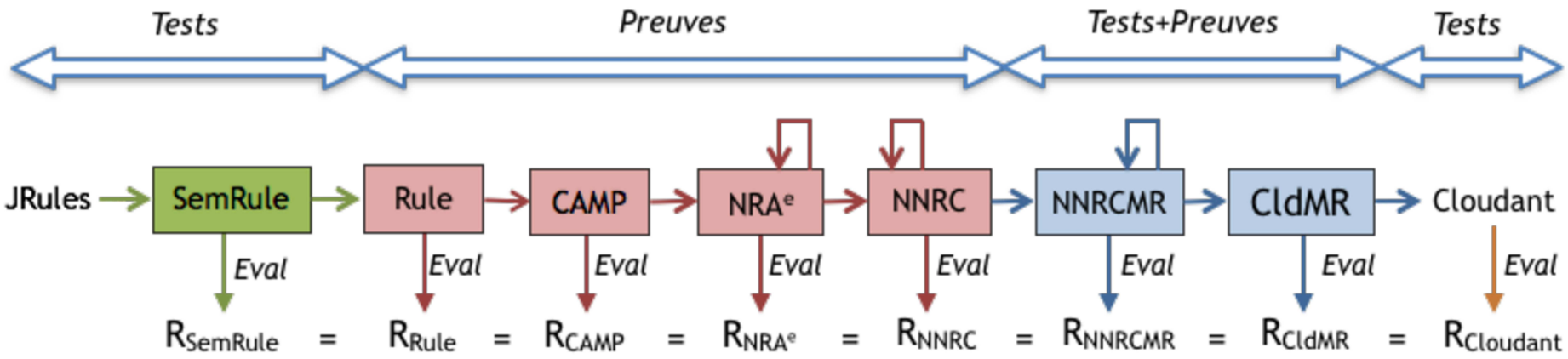
Choix entre Preuves et tests




```
Definition nraenv_eval c (e:nraenv) (env:data) (x:data)
  : option data := ...
```

- JRules & Cloudant : sémantique définie par leur implantation
- Cœur du compilateur : preuve de preservation de la sémantique (optimiseurs inclus)
- Compilateur vers Map/Reduce : utilise le test, preuves de quelques propriétés :
  - Chaîne de map/reduce bien formée (DAG)
  - Correction des réécritures (seulement preuve locale)

Choix entre Preuves et tests



Remarques :

- Cœur du compiler basé sur des langages connus (et relativement bien formalisés)
- Compilation vers map/reduce à partir de zéro : preuves rendaient le travail pénible
- Deboguage grandement facilité par le cœur prouvé (élimine énormément de code en cas de bogue)

Database Systems: The Complete Book, Ullman

16.2.2 Laws Involving Selection

. . . . . .

To start, when the condition of a selection is complex (i.e., it involves conditions connected by AND or OR), it helps to break the condition into its constituent parts. The motivation is that one part, involving fewer attributes than the whole condition, may be moved to a convenient place where the entire condition cannot be evaluated. Thus, our first two laws for σ are the *splitting laws*:

- $\sigma_{C_1 \text{ AND } C_2}(R) = \sigma_{C_1}\left(\sigma_{C_2}(R)\right)$.
- $\sigma_{C_1 \text{ OR } C_2}(R) = \left(\sigma_{C_1}(R)\right) \cup_S \left(\sigma_{C_2}(R)\right)$.

```
(* Relational rewrite: σ( q₂ ∧ q₁ )( q ) = σ( q₁ )(σ( q₂ )( q ))
   Notes:
    - This rewrite is only true in the absence of
      failure (i.e., for well-typed queries) *)

Lemma selection_split_and (q q₁ q₂:algenv) :
  σ( q₂ ∧ q₁ )(q) ⇒ σ( q₁ )(σ( q₂ )(q)).
Proof.
  ...
Qed.

(* Relational rewrite: σ( q₂ ∨ q₁ )( q ) = σ( q₁ )( q ) ∪ σ( q₂ )( q )
   Notes:
    - Over bags rather than sets, this is true for
      'maximal union', but not for 'additive union' *)

Lemma selection_split_or (q q₁ q₂:algenv) :
  σ( q₂ ∨ q₁ )( q ) ⇒ σ( q₁ )( q ) ∪max σ( q₂ )( q ).
Proof.
  ...
Qed.
```

Optimisations Algébriques

```
(**********
 * Complex *
 **********)

(* ♯flatten(χᵉ(χ⟨ENV⟩(σ⟨ q₁ ⟩(`{| ID |}))))) ∘ᵉ χ⟨ENV⟩(σ⟨ q₂ ⟩(`{| ID |}))
     ⇒ χ⟨ ENV ⟩(σ⟨ q₁ ⟩(σ⟨ q₂ ⟩(`{| ID |})) *)

Lemma tcompose_selects_in_mapenv_arrow q₁ q₂ :
  (♯flatten(ANMapEnv (χ⟨ENV⟩(σ⟨ q₁ ⟩( `{| ID |}))))) ∘ₑ (χ⟨ENV⟩(σ⟨ q₂ ⟩( `{| ID |})))
                                                    ⇒ (χ⟨ENV⟩(σ⟨ q₁ ⟩(σ⟨ q₂ ⟩( `{| ID |})))).
Proof.
  apply (rewrites_typed_with_untyped _ _ (compose_selects_in_mapenv q₁ q₂)).
  intros; algenv_inferer.
Qed.

(* (χᵉ⟨ q ⟩) ∘ᵉ (ENV ⊗ [ a : ID ]) ⇒ χ⟨ (q ∘ ENV·a) ∘ᵉ ID ⟩(ENV ⊗ [ a : ID ]) *)

Lemma tappenv_mapenv_to_map_arrow q a:
  ANAppEnv (ANMapEnv q) (ENV ⊗ `[| (a, ID)|]) ⇒
           χ⟨(q ∘ (ANUnop (ADot a) ANEnv)) ∘ₑ ID⟩( (ENV ⊗ `[| (a, ID)|]) ).
Proof.
  unfold talgenv_rewrites_to; intros; simpl.
  algenv_inferer.
  econstructor; eauto.
```

Remarques :

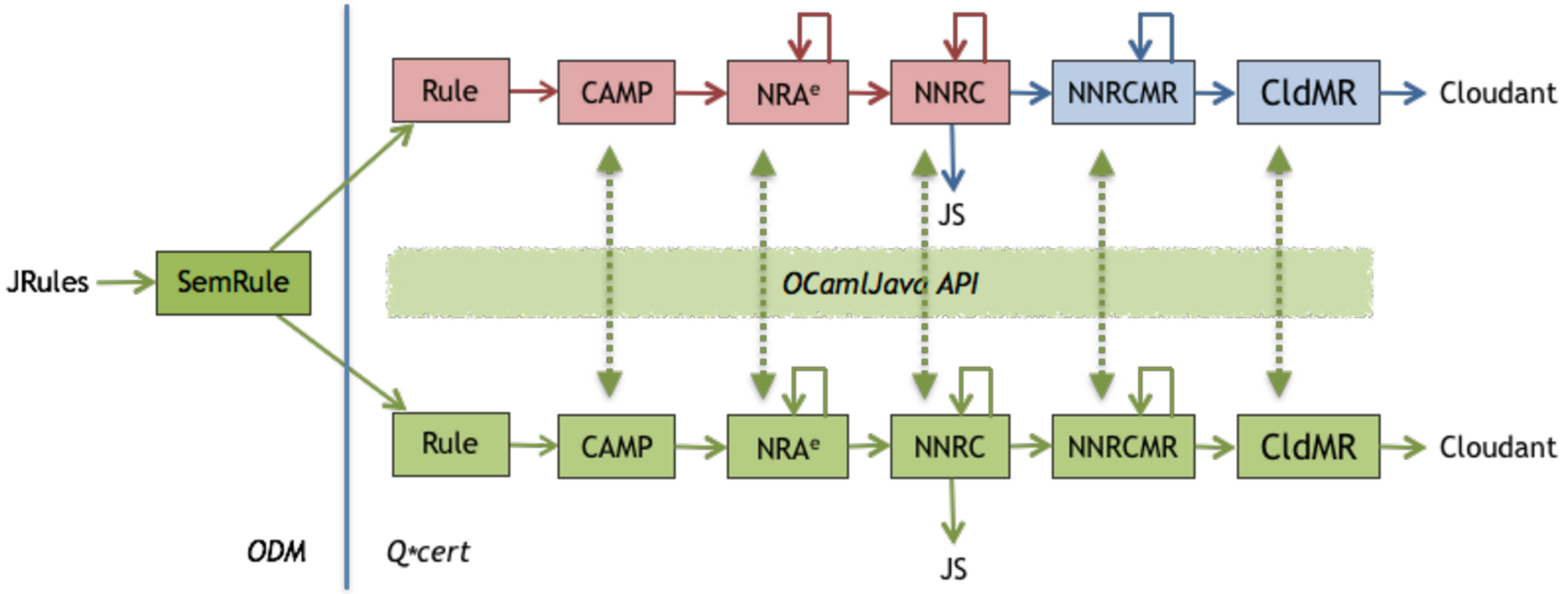
- Optimiseur contient environ 100 réécritures, la plupart non classiques (dans un sens bases de données)
- Difficile de prouver les optimisations complexes sans outil

Portage vers Java

Objectifs :

- Code java 'idiomatique' qui peut etre compris et maintenu par l'équipe produit
- Éviter (ou identifier) les divergences entre le code Coq et Java
- Assurer que le comportement des deux implantations est similaire

Portage : Double chaine Coq-Java



- Coq extrait vers OCaml puis jar avec OCamlJava
- Double hélice permet une combinaison arbitraire Coq/Java
- Tests traductions = comparaisons entre ASTs
- Tests optimisations = comparisons entre traces
- Portage environ trois/quatre semaines

Portage : Code

```
/** From TOptimEnvFunc.v: last checked 5/2/2016
     Definition tselect_and_fun {fruntime:foreign_runtime} (p: algenv)
      := match p with
           ANSelect op1 (ANSelect op2 op) =>
           ANSelect (ANBinop AAnd op2 op1) op
         | _ => p end. */
private static class tselect_and_fun implements OptFun {
  public NraNode optimize(NraNode nra) {
    if (nra instanceof NraSelect) {
      NraNode op1 = nra.getOperand1();
      NraNode select = nra.getOperand2();
      if (select instanceof NraSelect) {
        NraNode op2 = select.getOperand1();
        NraNode op = select.getOperand2();
        return new NraSelect(
         new NraBinaryOperator(BinaryOperator.And, op2, op1), op);
      } }
    return nra; } }
```

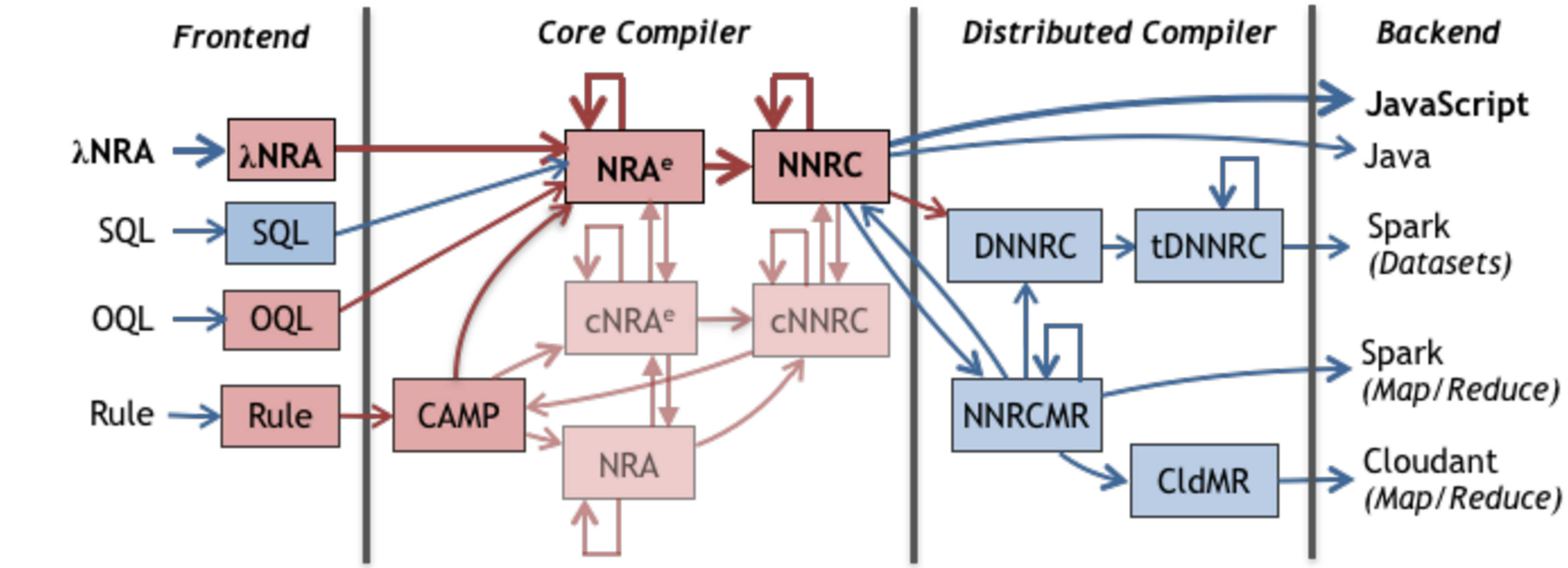
Coq : Souhaits

- Type Classes (pour les types externes) : complexité d'identifier le contexte nécéssaire dans une module (et messages d'erreur)
- Notations : source de frustration dans l'équipe (et copier coller vers LaTeX)
- Refactoring : renommage de modules, de types, de fonctions (à travers les notations, preuves, etc)
- Deboguer avec autre chose que les preuves ou `Eval`
- `Qed` vs `Defined` : Identifier les parties du code qui sont opaques

Parties non triviales ou innovantes

- Système de type à objets pour langages sur les données (méthode : branded values/types), typage et inférence de type, preuves correspondantes [Wadlerfest'2016]
- Gestion de l'environnement/variables dans les langages intermédiaires. NRA^e à base de combinateurs et preuve que les équivalences dans NRA s'appliquent à NRA^e [SIGMOD'2017]
- Modèle pour introduire la distribution (map/reduce)
- `CompilerDriver` pour gérer les chemins de compilations et les options correspondantes

Q*cert



λNRA = NRA with Lambda Terms
CAMP = Calculus of Aggregating Matching Patterns
Rule = Rule Macros for CAMP

NRA = Nested Relational Algebra
NRAe = NRA with Environments
cNRAe = Core NRAe
NNRC = Named Nested Relational Calculus
cNNRC = Core NNRC

DNNRC = Distributed NNRC
tDNNRC = Typed DNNRC
NNRCMR = NNRC with Map/Reduce
CldMR = NNRC with Cloudant Map/Reduce

https://querycert.github.io

Conclusion

- Coq pour le prototypage (plutôt que pour certifier)
- Coût de développement supplémentaire peut être justifié dans certains cas
- Pour de gros projets, avoir une partie du code vérifiée peut réduire de façon importante les coûts de debogue
- Bonne surprise : ajouter SQL au compilateur en 6 semaines (grâce à des langages intermédiaires bien défini)
- Un compilateur de requêtes réellement certifié est envisageable