Verification of the SQL compilation chain

Chantal Keller
Joint work with Léo Andres, Véronique Benzaken, Évelyne Contejean, Raphaël Cornet and Eunice Martins

January, 23rd 2018

UNIVERSITÉ PARIS SUD | FACULTÉ DES SCIENCES D'ORSAY
université PARIS-SACLAY

Formal guaranties for data-centric applications

Relational databases

well-studied theory [Codd70]

mature implementations

Oracle, DB_2 IBM, SQLServer, Postgresql, MySql, SQLite ...

a standard SQL

SQL: a declarative language

say what but not how

```
select lastname
from people p, director d, role r, movie m
where
  d.mid = r.mid and d.pid = r.pid and p.pid = d.pid and m.mid = d.mid
  and m.year > 1950;
```

SQL: a declarative language

say what but not how

```sql
select lastname
from people p, director d, role r, movie m
where
  d.mid = r.mid and d.pid = r.pid and p.pid = d.pid and m.mid = d.mid
  and m.year > 1950;
```

1. collect all the data, then filter

SQL: a declarative language

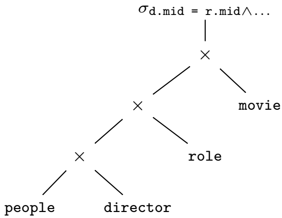
say what but not how

```
select lastname
from people p, director d, role r, movie m
where
  d.mid = r.mid and d.pid = r.pid and p.pid = d.pid and m.mid = d.mid
  and m.year > 1950;
```

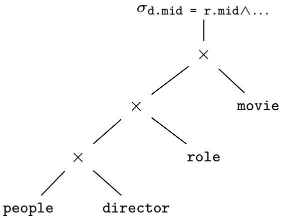
1. collect all the data, then filter
2. collect only useful data, as fast as possible

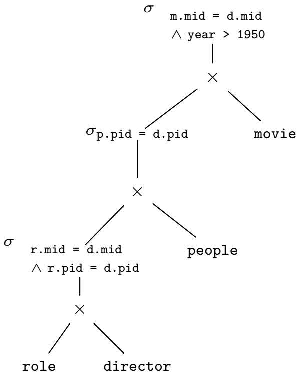
Compilation: from solution 1 to solution 2

```sql
select lastname
from people p, director d, role r, movie m
where
  d.mid = r.mid and d.pid = r.pid and p.pid = d.pid and m.mid = d.mid
  and m.year > 1950;
```

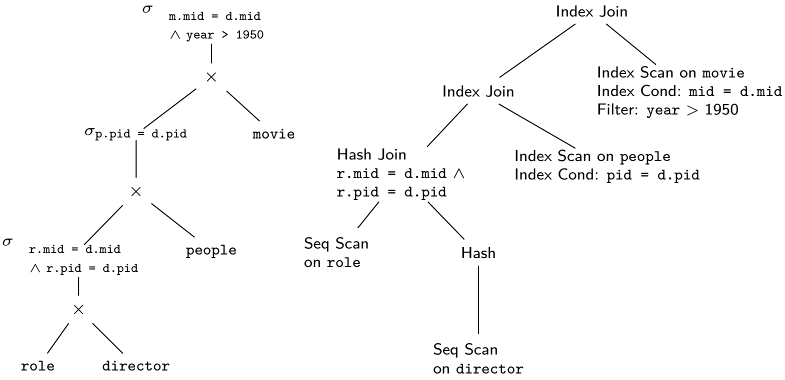


Compilation: from solution 1 to solution 2

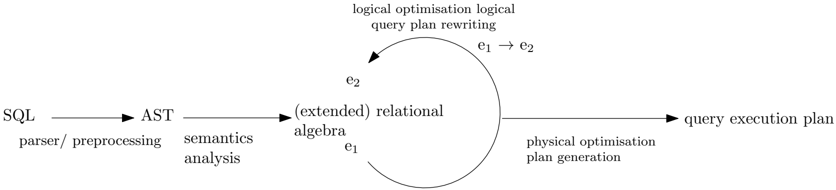




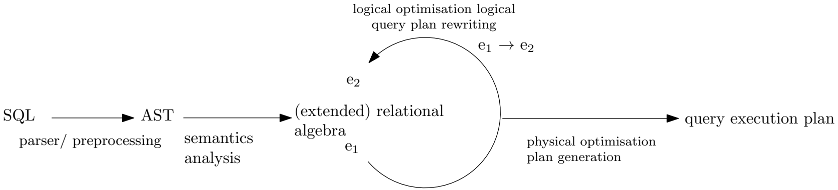
Compilation: from solution 1 to solution 2



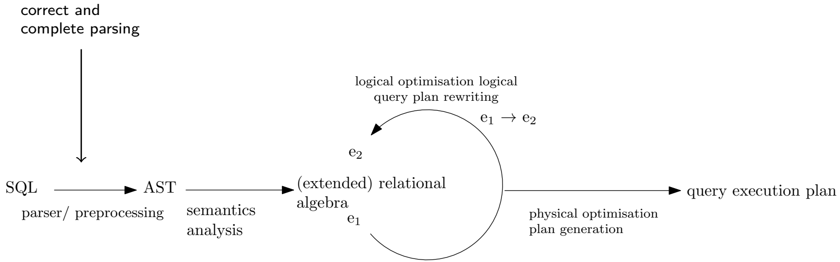
The compilation chain



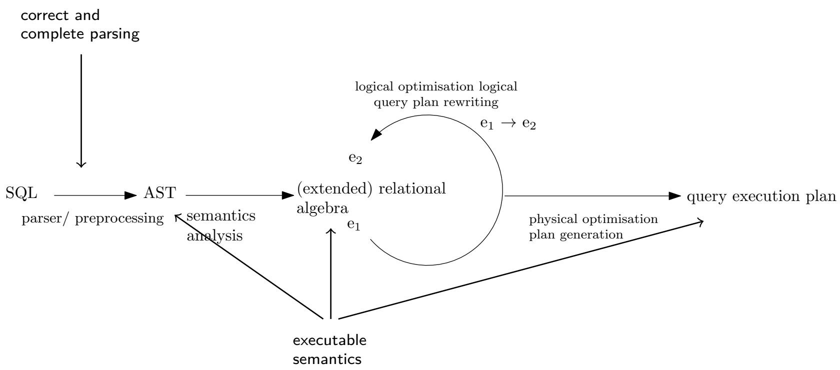
The compilation chain, verified in Coq



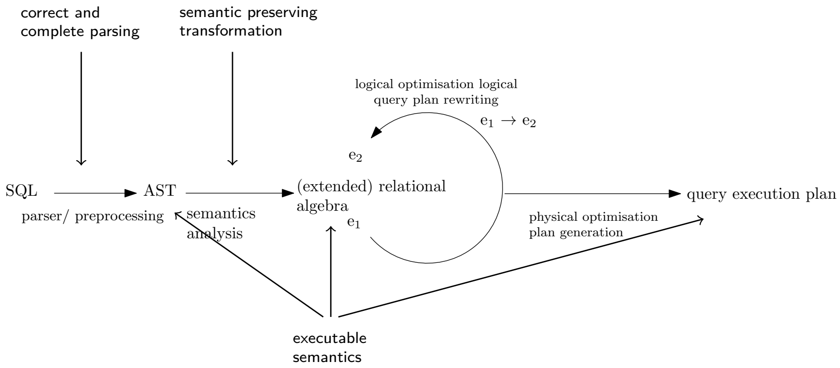
The compilation chain, verified in Coq



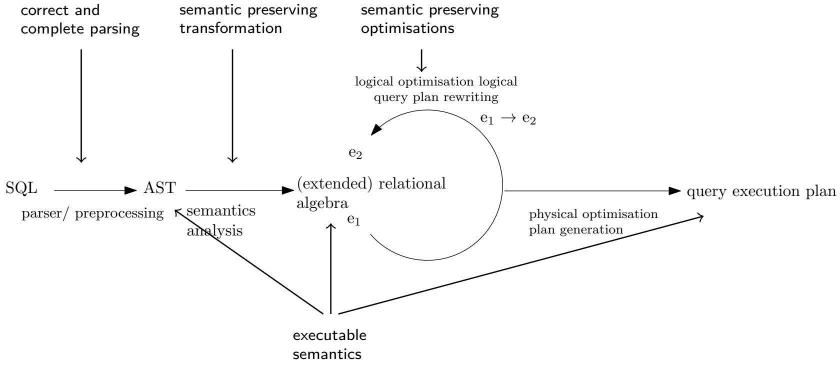
The compilation chain, verified in Coq



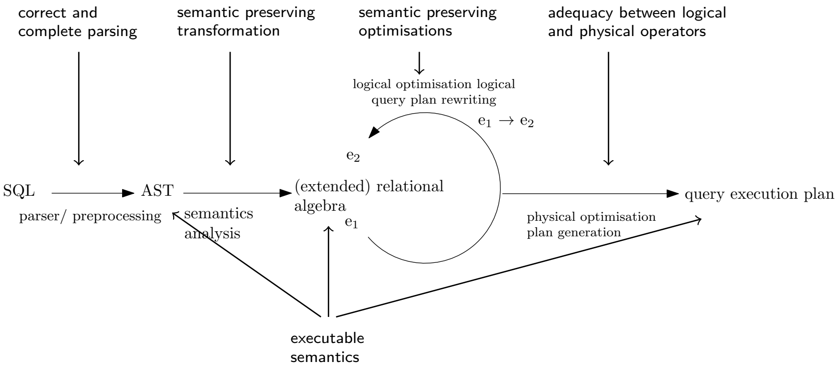
The compilation chain, verified in Coq



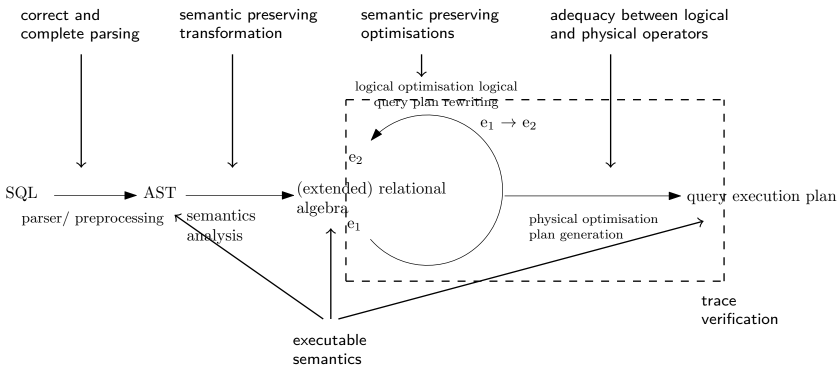
The compilation chain, verified in Coq



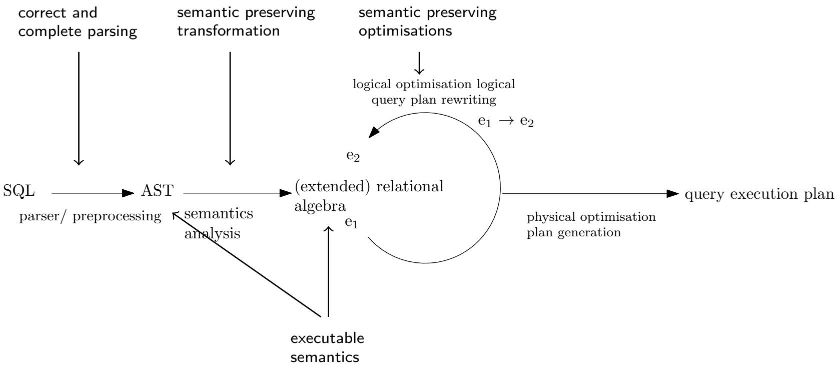
The compilation chain, verified in Coq



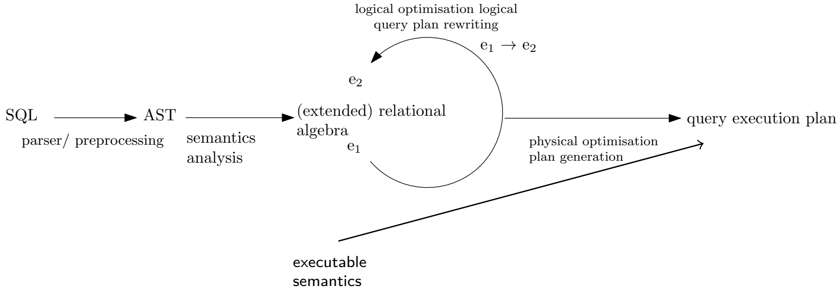
The compilation chain, verified in Coq



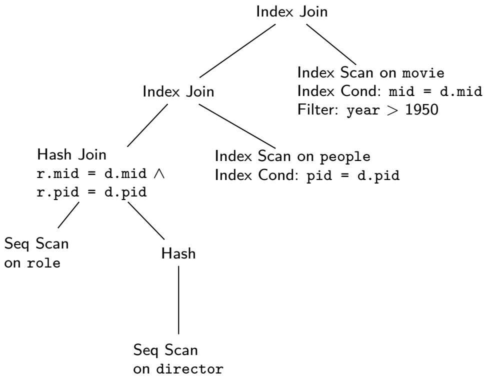
Not detailed parts



Runtime: physical algebra



Iterator interface: online algorithms



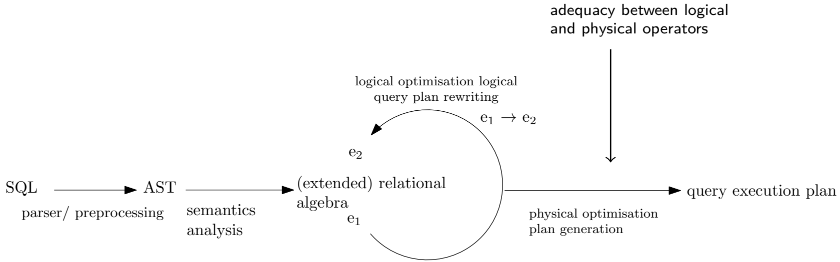
Abstract iterator interface

```
Record Cursor (elt : Type) : Type := {
  cursor : Type;
  next : cursor  → result elt * cursor;
  has_next : cursor  → Prop;
  reset : cursor  → cursor;

  collection : cursor  → list elt;
  visited : cursor  → list elt;
  coherent : cursor  → Prop;

  next_collection : forall c, coherent c  → collection (snd (next c)) = collection c;
  next_coherent : forall c, coherent c  → coherent (snd (next c));
  ...

  ubound : cursor  → nat;
  ubound_complete :
    forall c acc, coherent c  → ∼has_next (fst (iter next (ubound c) c acc));
 }.
```

Implementations and combinations

Iterator interface operators				
	ϕ algebra			
data centric operators	**simple**	**index based**	**sort based**	**sql algebra**
map	Seq Scan	Index scan Bitmap index scan	Sort scan	r, π
join	Nested loop Block nested loop	Hash join Index join	Sort merge join	$\times$
filter	Filter			σ
group	Group			γ
avg count max sum	Aggregate Hash Hash aggregate			aggregate
	Intermediate results storage operators			
	Materialize			

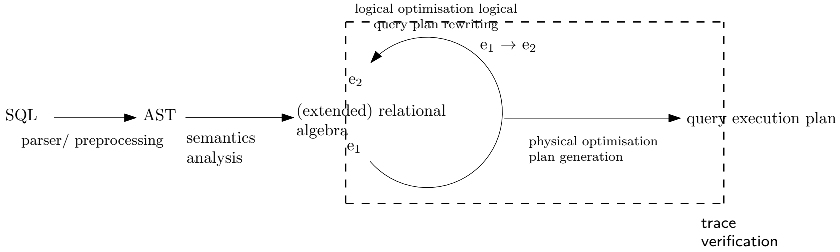
Adequacy between logical and physical operators



High-level specification of data-centric operators

```
Definition is_a_filter_op (init res : container) (p : A → bool) :=
  forall t, nb_occ t res = (nb_occ t init) * (if p t then 1 else 0).

Lemma Filter_is_a_filter_op c p:
  is_a_filter_op (materialize c) (materialize (mk_Filter c p)) p.

Lemma Sigma_is_computable_by_any_filter_op :
    forall init res p, is_a_filter_op init res p →
    forall q,
      eval_query env q =R= content init →
        eval_query env (Sigma p q) =R= content res.
```

Certification of query execution plans



Using traces

A good query planner is complex software

Instead, call an external planner and check the answer

Oracle, Postgresql

Bonus: *a posteriori* certification of these software

Checking the answer

We adopt a reflexive approach

1. associate a logical operator to each physical operator: adequacy
2. guess which optimisations have been used: “replay” them using the property that they are semantic preserving

Guessing may be hard: no known normal form

Conclusion

The first verification of the full SQL compilation chain

Combines lots of techniques provided by Type Theory

- high degree of abstraction
- "traditional" proofs of algorithms
- reflexive verification of traces
- program extraction
- ...

Perspective: non-relational databases
abstraction should be good enough to handle them