Formalisation of Relational Algebra and a SQL-like Language with the RISCAL Model Checker

Joachim Borya

Johannes Kepler Universität

March 28th, 2023

Outline

Goals

Our essential goals are:

- Formalization of **Relational Algebra** (RA) and a SQL-like **Query Language** using RISCAL (as presented on January 26th)
- Verification of **RA operations** and **theorems** using **RISCAL**, **SMT solvers**, and **RISCTP**
- Benchmarking the verification process (today)

Setup

- **Host system**
  - **OS:** Windows 10 Version 22H2
  - **CPU:** Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz (2 cores)
  - **RAM:** 16000 MB
- **Virtual Machine**
  - **OS:** Debian GNU/Linux 11 (bullseye)
  - **RAM:** 11370 MB
- **Java:** openjdk 11.0.18 2023-01-17
- **RISCAL:** Version 4.2.2 (September 16, 2022)

RISCAL was executed with 4 threads.

Methods

- **RISCAL Bounded Model Checker** (exhaustive testing) and **SMT solvers** for models with fixed parameters $M = N = 2$
- **RISCTP Proving Interface** for models with arbitrary parameters M, N

Results

	RISCAL	TP (Z3)	Yices	Z3	Boolector	CVC4
rUnion	52	5473	97	115	36	72
rIntersect	41	5341	49	23	40	59
rMinus	45	5180	37	22	8	39
concat	247	364	12	67	1119	269
cartesian	467	395	54	1490	36095	26230
select	92	323	52	64	1210	661
project	97	560	43	122	27419	2965
join	86	386	88	3448	191882	68600

Table 1: Benchmark Operations (ms)

- SMT-LIB translation and subsequent check is generally the fastest method.
- RISCAL performs very well in some cases (see join).
- Yices is the best performing SMT solver.

Results

	RISCAL	TP (Z3)	Yices	Z3	Boolector	CVC4
select_union_equiv	112	289	7	53	30	107
select_intersect_equiv	187	285	31	54	9	85
select_minus_equiv	182	251	53	22	35	108
select_intersect_comp	67	294	30	38	38	98
join_cartesian_subset	83	4109	67	4285	167761	107472

Table 2: Benchmark Theorems (ms)

- Theorem checking is very fast with SMT solvers, especially Yices.
- TP is also a fast option, despite accomplishing more than SMT.

Data

0	
0	**1**
0	0
0	1

1	
0	**1**
0	1
1	0
1	1

Table 3: Sample database

Query Executions

```
var q1:Query := Query!select(List!node(2, List!node(0, List!nil)), Query!where(1, 0,
     Query!on(0, 1, Query!from(0), Query!from(1))));

var q2:Query := Query!select(l, Query!where(1, 1, Query!from(0)));

var q3:Query := Query!select(List!node(1, List!node(2, List!nil)), Query!on(0, 1, Query!
     from(0), Query!from(1)));

var q4:Query := Query!select(l, Query!from(1));

var q5:Query := Query!where(1, 0, Query!on(0, 1, Query!from(0), Query!from(1)));

var q6:Query := Query!where(1, 0, Query!from(0));

var q7:Query := Query!on(0, 1, Query!from(0), Query!from(1));

var q8:Query := Query!from(0);
```

Query Executions

```
Query result:
[2,{[1,0,0,0]}]
SELECT
2,
0
FROM 0
JOIN 1
ON 0 = 1
WHERE 1 = 0
;

Query result:
[2,{[1,0,0,0]}]
SELECT
1,
0
FROM 0
WHERE 1 = 1
;

Query result:
[2,{[0,1,0,0],[1,1,0,0]}]
SELECT
1,
2
FROM 0
JOIN 1
ON 0 = 1
;
```

```
Query result:
[2,{[1,0,0,0],[1,1,0,0],
[0,1,0,0]}]
SELECT
1,
0
FROM 1
;

Query result:
[4,{[0,0,1,0]}]
SELECT *
FROM 0
JOIN 1
ON 0 = 1
WHERE 1 = 0
;

Query result:
[2,{[0,0,0,0]}]
SELECT *
FROM 0
WHERE 1 = 0
;
```

```
Query result:
[4,{[0,0,1,0],[0,1,1,0]}]
SELECT *
FROM 0
JOIN 1
ON 0 = 1
;

Query result:
[2,{[0,0,0,0],[0,1,0,0]}]
SELECT *
FROM 0
;
```

Final steps

To do: Refine the text.

Scheduled finalization: After the Easter break.

Thank you for your attention!