SRA OSS, INC.

Extending View Updatability by a Novel Theory

Prototype Implementation on PostgreSQL

Yugo Nagata, Yoshifumi Masunaga

PGCon2017
2017.5.25

Outline

- Introduction
  - View update problem
- Novel theory
  - The intention-based approach
- Prototype Implementation
  - Demo
- Summary

What is “view”?

- Virtual relation based on the result-set of a stored query

- Purposes of views
  - Simplification of complex queries that are used repeatedly
  - Logical data independence
  - Database security

View Examples: Selection Views

K41-EMP

ENO	ENAME	DNO	SAL
003	J. Smith	K41	40
006	R. Jones	K41	20

POOR-EMP

ENO	ENAME	DNO	SAL
006	R. Jones	K41	20

Virtual

CREATE VIEW K41-EMP AS
SELECT * FROM EMP
WHERE DNO = 'K41'

CREATE VIEW POOR-EMP AS
SELECT * FROM EMP
WHERE SAL =< 30

EMP[DNO = 'K41']

EMP[SAL ≦ 30]

EMP

ENO	ENAME	DNO	SAL
003	J. Smith	K41	40
006	R. Jones	K41	20
007	J. Bond	K55	60

Real

View Update problem

- Users want to access views as real relations
  - Read queries:
    - SELECT … always OK
  - Update queries:
    - DELETE, INSERT, UPDATE

- Views are not always updatable
  - What kind of views are updatable?
  - How to translate update requests on views into requests on base relations?

Translation ambiguity

- Natural join view V from R and S

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

- Request to delete (a, b', c) from V
- Three translation alternatives
  - T1: delete (a, b') from R
  - T2: delete (b', c) from S
  - T3: execute both T1 and T2

Translation ambiguity cannot be resolved.

→ This view is not updatable

Updatable views in PostgreSQL

- Simple views are automatically updatable:
  - Exactly only one table or another updatable view in its FROM list
  - Not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  - Not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  - Select list must not contain any aggregates, window functions or set-returning functions.

SQL Standard

- PostgreSQL's updatable view is basically according to SQL-92
- View updatability is extended at SQL:1999
  - JOIN and UNION ALL views are updatable under some conditions.
  - Oracle supports it partially

INSTEAD OF triggers resolve all?

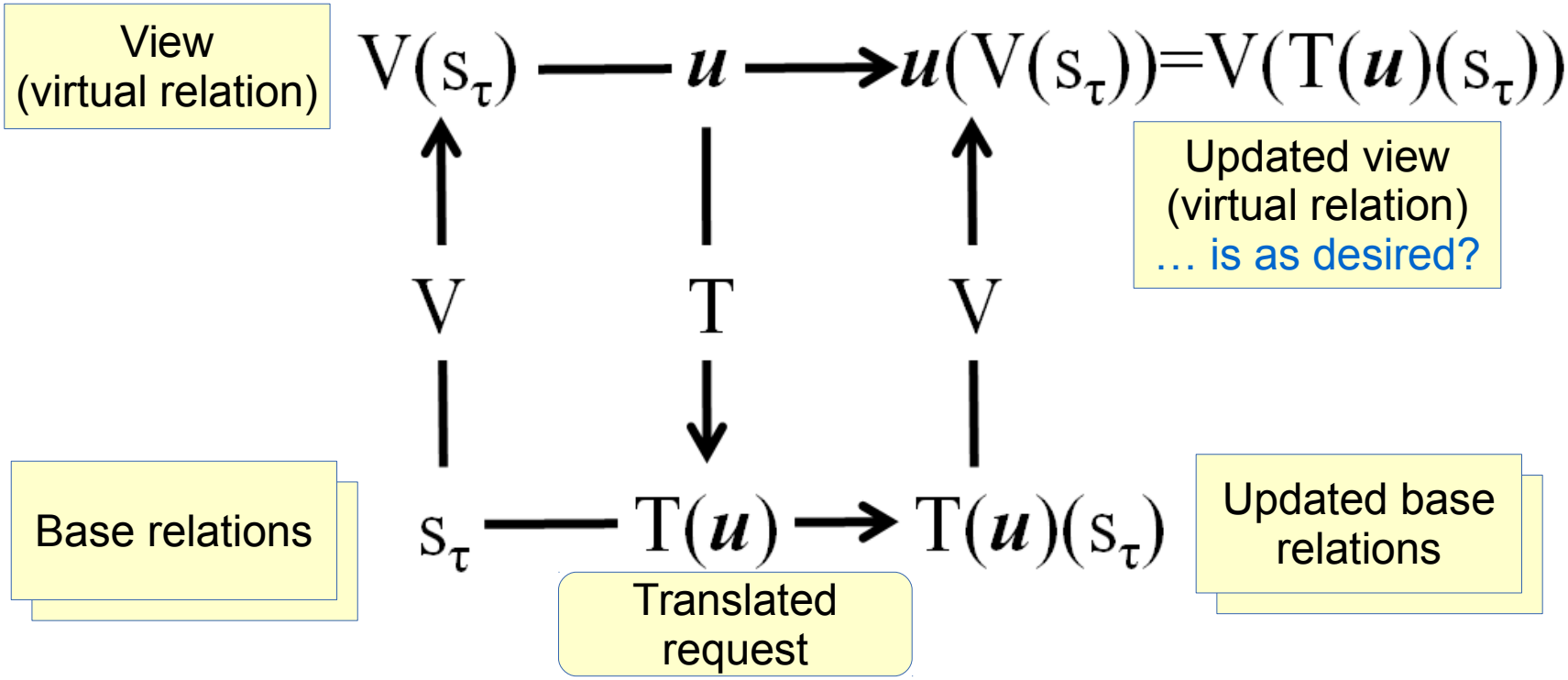
- Create INSTEAD OF triggers on views
  - User can do any actions in the trigger functions
  - Convert the attempted query into appropriate actions on other table

- Specific triggers need to be defined for each views.
- The best way of using it is not clear.

Novel Theory:
An Intention-based approach

View Updatability

V : View definition
u : Update request for view
s_τ : State of base relations
T : Translation of requests



The view is updatable if we can find an translation T that has ***no side effects, and is unique.***

Traditional approaches

- Syntax-based /Functional Approach
  - Dayal et. al. [‘79] and many others
- Semantics-based Approach
  - Masunaga [‘84], Keller[‘86], and others
- Interaction-based Approach
  - Sheth et. al.[‘88]

Not Yet Fully Resolved.
An Old and New Problem.

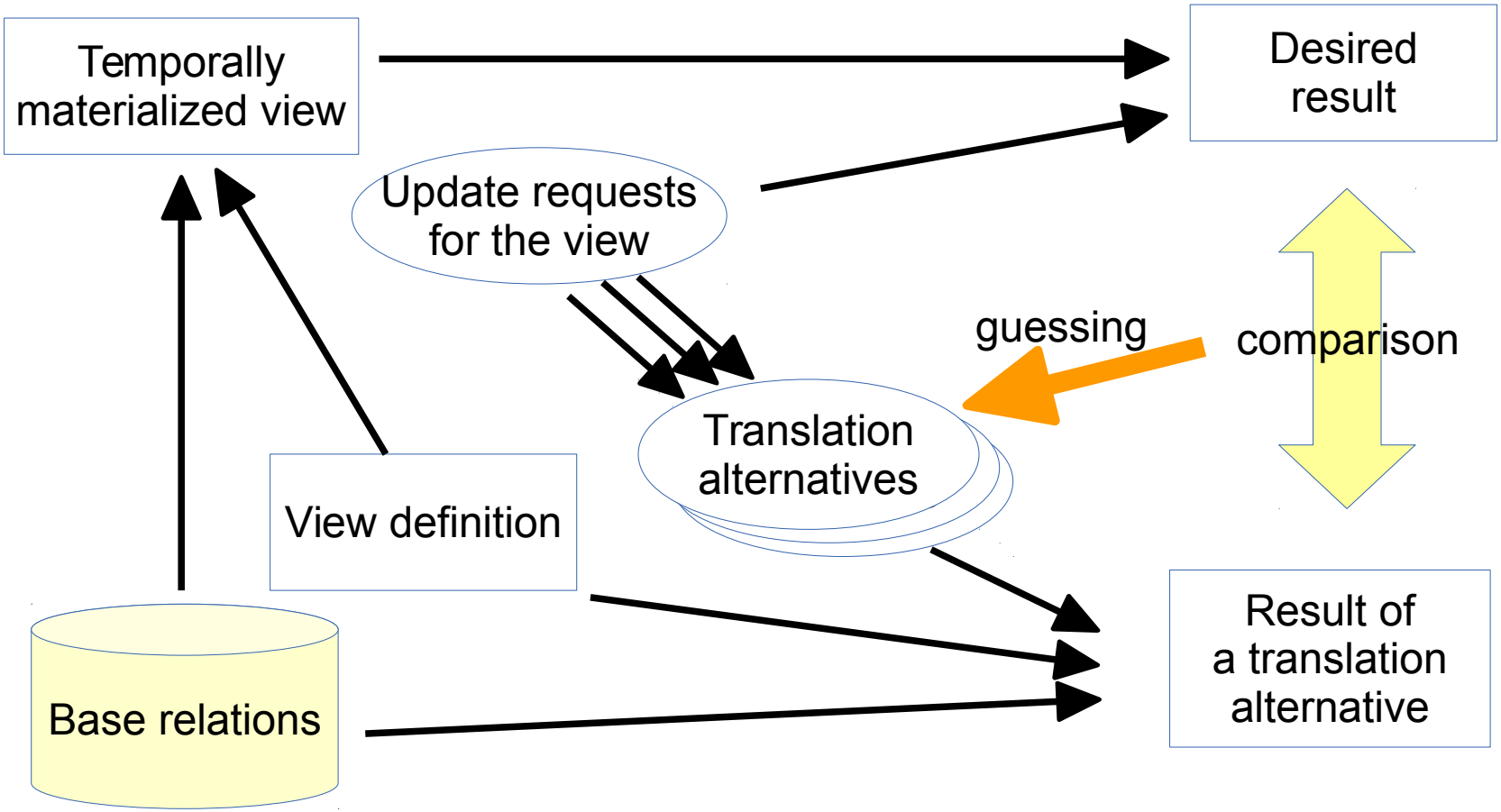
The Intention-based Approach

- Resolving view update translation ambiguity by guessing the user's INTENTION of update

"View updatability based on pro forma guessing of update intention"

- Idea:
  - Compute each translation candidate "temporarily" so that we can "guess" the user's update intention uniquely
  - It depends on data in base relations.

View updatability based on pro forma guessing of update intention



Example of pro forma guessing of update intention (1)

- Natural join view V from R and S

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

- Request to delete {(a, b, c), (a,b,c')} from V
- Three translation alternatives
  - T1: delete (a, b) from R
  - T2: delete {(b, c), (b, c')} from S
  - T3: execute both T1 and T2

Only T1 can realize the desired result without side effects.

→ T1 is the user's update intention!

Example of pro forma guessing of update intention (2)

- Natural join view V from R and S

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

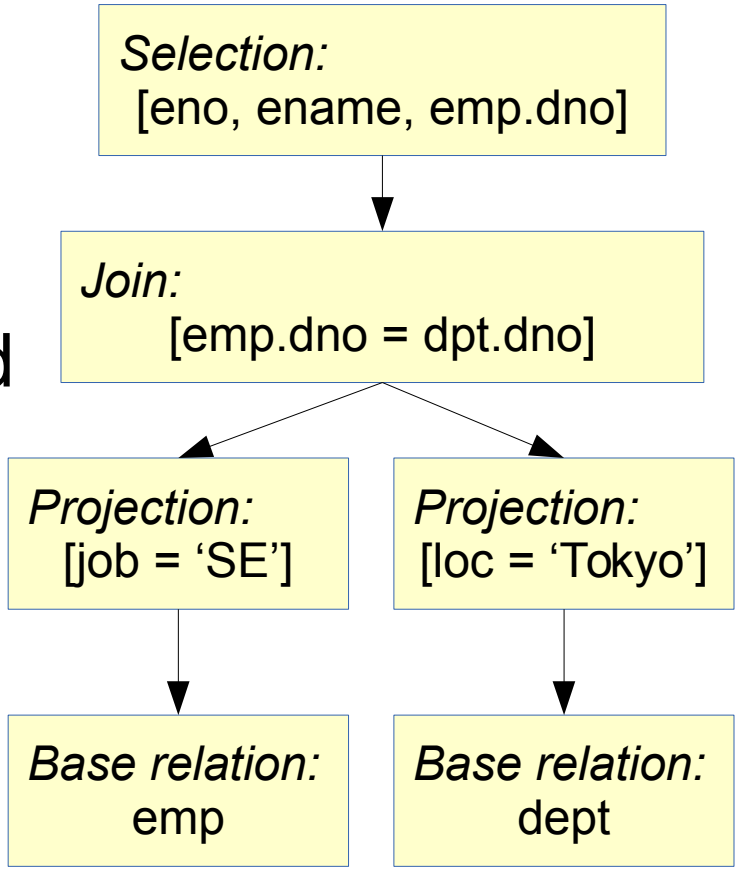
- Request to delete (a, b', c) from V
- Three translation alternatives
  - T1: delete (a, b') from R
  - T2: delete (b', c) from S
  - T3: execute both T1 and T2

All of T1,T2 and T3 realize the desired result.

→ we cannot guess the update intention!

Updatability of generally defined views

- A View is defined recursively using base relations and predefined views.
  - View definition tree
  - View updatability can be checked recursively.




```
CREATE VIEW emp_se_tokyo AS
 SELECT eno, ename, emp.dno
 FROM emp JOIN dept ON emp.dno = dpt.dno
 WHERE job = 'SE' AND loc = 'Tokyo'
```

Implementing a prototype on PostgreSQL

Prototype implementation on PostgreSQL

- As a Proof of Concept
  - To test feasibility of the theory

- The prototype is developed as an EXTENSION of PostgreSQL.

Two approaches

- Rule-based approach
  - View support is realized with the rule system in PostgreSQL
  - Update requests is provided as Query tree
  - Need to modify PostgreSQL's core code.

- Trigger-based approach
  - Implement the algorithm in trigger functions
  - Update requests is provided as a list of tuples

  - We decided to start with the trigger-based approach

Problem of INSTEAD OF triggers

- “STATEMENT LEVEL” INSTEAD OF trigger is not supported
  - Only “ROW LEVEL” is supported
  - We need to process multiple tuples to translate requests.

- Use STATEMENT LEVEL AFTER trigger instead of INSTEAD OF trigger

Transition table

- Transition table
  - A new feature of PostgreSQL 10

> • Add AFTER trigger transition table to record changed rows (Kevin Grittner)
>
> Transition table contents are accessible from server-side languages.

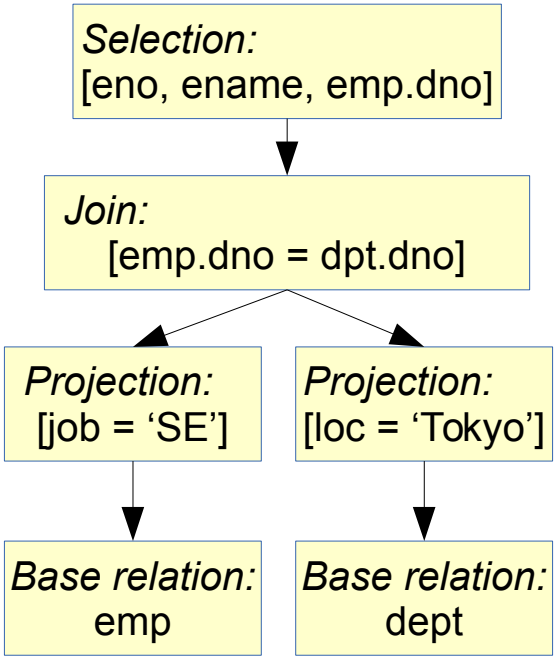
  - The before or after images for rows affected by the statement which fired the trigger can be accessed as tuplestores in AFTER trigger functions.
  - We use tuples in these tuplestores as the requests for views.

Overview of the implementation

- In BEFORE trigger
  - Build a view definition tree
- In INSTEAD OF trigger
  - Do nothing
  - Necessary to suppress auto-updatable view.
- In AFTER trigger
  - Extract the request for the view from transition tables
  - Check the view updatability and update the base relations if possible.

Building view definition trees

- Convert Query tree into a view definition tree
  - The Query tree of view definition is available by get_view_query()
  - We need to convert any sub tree to SQL of view definition
    - To check the view updatability recursively, sub tree need to be temporally materialized.



Update requests for views

- Update requests for views are represented as lists of tuples

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

  - `DELETE FROM v WHERE C = c'`

    → delete {(a,b,c'), (a',b,c')}
  - `UPDATE V SET C = d WHERE C = c'`

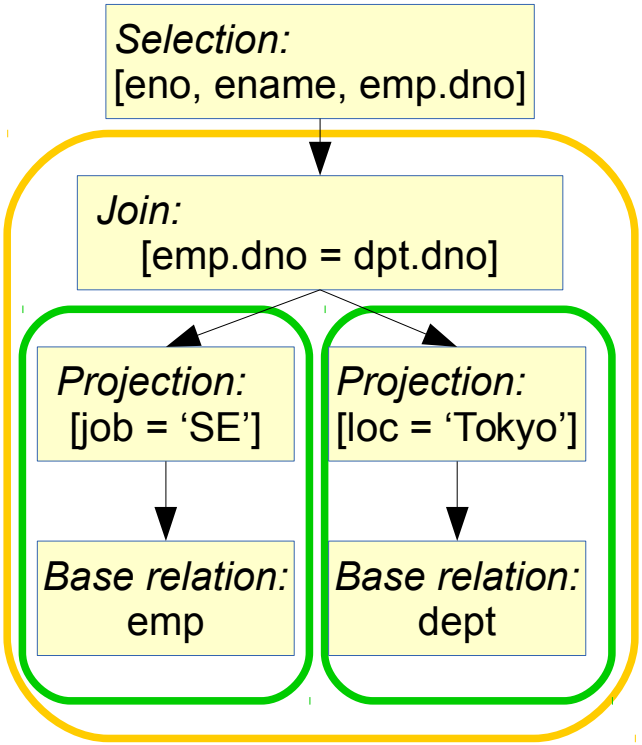
    → rewrite {(a,b,c'), (a',b,c')} to {(a,b,d), (a',b,d)}
  - `INSERT INTO V VALUES (a, b', d), (a, b', e)`

    → insert {(a,b',d), (a,b',e)}

- The lists are extracted from the tuplestores, `tg_oldtable` and `tg_newtable` in `Trigger` structure, in the AFTER trigger function.

Check view updatability

- Walk down from the top of view definition tree recursively
  - If we find a join node, use “Pro forma guessing” algorithm.
  - The sub-tree of the join node and the children node are temporally materialized.
    (temporary tables are created)



Pro forma guessing

- The update requests are divided for base relations.

  Insert {(a,b',d), (a,b',e)} into V

  → insert (a,b') into R
  insert {(b',d), (b',e)} into S

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

- Generate alternatives of translation
  - The alternatives are determined logically.

1. Insert {(b',d),(b',e)} into S
2. Insert (b',d) into S
3. Insert (b',e) into S
4. Insert (a,b') into R
5. Insert (a,b') into R, insert {(b',d),(b',e)} into S
6. Insert (a,b') into R, (b',d) into S
7. Insert (a,b') into R, insert (b',e) into S
8. Insert (a,b'),(a,b') into R Insert {(b',d),(b',e)} into S
9. Insert (a,b'),(a,b') into R, Insert (b',d) into S
10. Insert (a,b'),(a,b') into R, Insert (b',e) into S

- Check if only one of these alternatives can realize the desired result.
  → The join view is updatable and the translation is the answer!

Example (1)

```
- INSERT INTO base1 VALUES (1,1),(2,1),(1,2);
- INSERT INTO base2 VALUES (1,2),(2,1);

- CREATE VIEW v(a,b,c) AS SELECT a,base1.b,c
    FROM base1 JOIN base2 ON base1.b = base2.b;

- INSERT INTO v VALUES (1,2,3),(1,2,4);
ERROR:  cannot insert into view "v"
DETAIL:  Views that do not select from a single table
or view are not automatically updatable.
HINT:  To enable inserting into the view, provide an
INSTEAD OF INSERT trigger or an unconditional ON INSERT
DO INSTEAD rule.

- SELECT add_view_ext('v');

- INSERT INTO v VALUES (1,2,3),(1,2,4);
INSERT 0 2
- INSERT INTO v VALUES (2,2,2);
ERROR:  not updatable based on pro forma guessing
```

(base1)

a	b
1	1
2	1
1	2

(base2)

b	c
1	2
2	1

a	b	c
1	1	2
2	1	2
1	2	1

a	b	c
1	1	2
2	1	2
1	2	1
1	2	3
1	2	4

(base1)

a	b
1	1
2	1
1	2

(base2)

b	c
1	2
2	1
2	3
2	4

Example (2)

```
- DELETE FROM v WHERE c in (1,3);
DELETE 2
- DELETE FROM v WHERE c = 4;
ERROR:  not updatable based on pro forma guessing

- UPDATE v SET c = 20 WHERE c = 2;
UPDATE 2
- UPDATE v SET c = 200 WHERE a = 2;
ERROR:  not updatable based on pro forma guessing
```

(base1)

a	b
1	1
2	1
1	2

(base2)

b	c
1	2
2	1
2	3
2	4

a	b	c
1	1	2
2	1	2
1	2	1
1	2	3
1	2	4

a	b	c
1	1	20
2	1	20
1	2	4

(base1)

a	b
1	1
2	1
1	2

(base2)

b	c
1	20
2	4

Demo

Conclusion

- View update problem
- The intention-based approach
  - Pro forma guessing of update intention
  - JOIN views are updatable in certain cases although they are not updatable in the traditional approach.
- Prototype Implementation
  - Trigger-based approach
  - Use transition table in AFTER trigger
    - A new feature of PostgreSQL 10

Future plans

- Handle the limitation and performance issues
- Test this prototype in many cases, and give feedback to the theory to elaboration it
- We might need to investigate the rule-based approach

Thank you