Introduction to Magic Sets

a.k.a. Sideways Information Passing (SIP)

Objective:

- Learn about the key *logical* optimization concerning
  - Nested queries (so also views)
  - Recursive Datalog programs and SQL queries (SQL 99)
  - It follows, central to graph database query engines, (but almost none are there yet).

Slide thanks: (originals can be found under readings)

- Ramakrishnan and Gehrke text slides
- Not borrowed for lecture, but are very goo, Stefan Brass's course slides

Scholarship:

With the identification of recursion in the relational model…

- In the beginning, invited paper, SIGMOD '86

  François Bancilhon, Raghu Ramakrishnan:
  An Amateur's Introduction to Recursive Query Processing Strategies. 16-52

Scholarship:

With the identification of recursion in the relational model…

And then (>1986), annually

- SIGMOD, VLDB, ICDE & PODS, at least one full session:

Recursion

- Isabel F. Cruz, Alberto O. Mendelzon, Peter T. Wood:
  A Graphical Query Language Supporting Recursion. 323-330
- H. V. Jagadish, Rakesh Agrawal, Linda Ness:
  A Study of Transitive Closure As a Recursion Mechanism. 331-344
- Weining Zhang, Clement T. Yu:
  A Necessary Condition for a Doubly Recursive Rule to be Equivalent tc

[SIGMOD '87]

Critical Papers:

The beginning of the end:

- 1990
  - (PODS) "Magic Conditions", Mumick et.al…. *Ramakrishnan*
  - (SIGMOD) "Magic is Relevant", same authors

Then:

- 1996
  - (SIGMOD) "Cost-Based Optimization for Magic: Algebra and Implementation", Seshadri, Hellerstein,…, *Ramakrishnan, …*

Commercial implementations followed, then standardized: SQL 99

- On going research – occasional paper, not >12 major papers a yesr

Two more things to know, upfront,
… to get your attention

- Conceptual, logical, physical….
  - Our interest, relative to graph database queries: *logical*
- Our physical targets are very different, but:

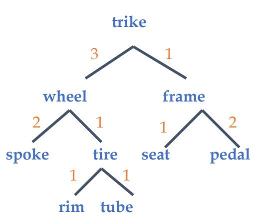
"**Pushing θ-Semijoin through Join:** We present below a transformation rule that describes how to push θ-semijoins through joins. [Cost-Based Optimization for Magic: Algebra and Implementation]

- Bloom filters:
  - Central to cloud-native data stores
  - Enable, fast, approximate, implementation of semi-join

Today's running example:

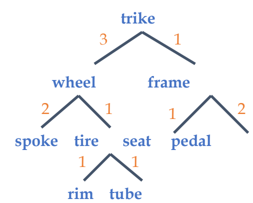
- From Ramakrishan and Gehrke

A Graph DB,
- directed, (not illustrated)
- edge labeled



part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	2
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

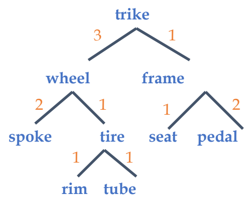
Assembly instance



- Find [all] components of "trike".
- Can you write a relational algebra query to compute answer on ***the given instance of Assembly***?

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance



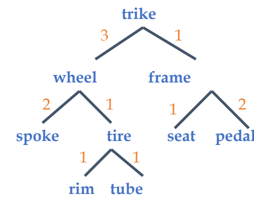
- Can determine, trike is made of wheels and frame
  - 1 self-join of Assembly
- Union, spoke, tire, seat and pedal
  - Starting with trike – two self joins
  - Each such self-join, extends the path by 1
- Union,
  - … need as many self-joins as height of the tree.
  - But, can't know tree height, without looking at the data

→ can't write the relational algebra before looking at the data.

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

9

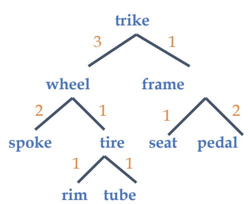


- All components of trike ≡ transitive closure of trike
- Can't express that in pure relational algebra

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

10

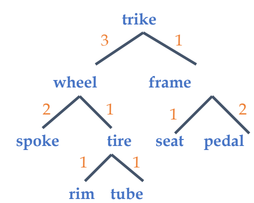


- Recall: relational algebra ≡ nonrecursive Datalog
- Unspecified path length → recursive Datalog

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

11



- Unspecified path length → recursive Datalog

Rule 1: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

Dependency graph: Comp

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

12

If not Relational Algebra, what about SQL?

In SQL

- Keyword, WITH
- A statement forms a *temporary, intermediate table,*
  - Statement is called a <Common Table Expression>, (CTE)
- As needed, keyword, RECURSIVE

If not Relational Algebra, what about SQL?

In SQL

- Keyword, WITH
- A statement forms a *temporary, intermediate table,*
  - Statement is called a <Common Table Expression>, (CTE)
- As needed, keyword, RECURISVE

Documenation:

- https://www.postgresql.org/docs/11/queries-with.html

Tutorials:

- https://www.sqlservertutorial.net/sql-server-basics/sql-server-cte/
- https://www.sqlservertutorial.net/sql-server-basics/sql-server-recursive-cte/

Recursive SQL Query Example

```
WITH RECURSIVE Comp(Part, Subpt) AS     /* Define Comp */

(SELECT A1.Part, A1.Subpt FROM Assembly A1)

UNION

(SELECT A2.Part, C1.Subpt
 FROM Assembly A2, Comp C1
 WHERE A2.Subpt=C1.Part)


SELECT * FROM Comp C2    /* Returns all parts of a trike */
```

15

Recursive SQL Query Example

```
WITH RECURSIVE Comp(Part, Subpt) AS     /* Define Comp */     Think of Comp is a View name

(SELECT A1.Part, A1.Subpt FROM Assembly A1)

UNION

(SELECT A2.Part, C1.Subpt
 FROM Assembly A2, Comp C1
 WHERE A2.Subpt=C1.Part)


SELECT * FROM Comp C2    /* Returns all parts of a trike */
```

1) Initialize: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).

2) Add through repeated, recursive, execution
Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt)

3) Get a result as is the net result, Comp, is the name of a view

16

Review: Revisit Naïve Evaluation

Beginning Cycle 1

Comp

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Comp, is an empty EDB table

Rule 2 is not satisfied

Eval rule 1, and get

Rule 1: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

18

End Cycle 1

Comp (1)

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Rule 1: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

19

Cycle 2

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Comp (1)

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Rule 1, **again:** Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Set theory – new rows don't appear twice

Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Comp tuples by applying Rule 2 once, (in cycle 2)

20

Cycle 3

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Comp tuples by applying Rule 2 again in cycle 3

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

—

Union, New, Comp tuples by applying Rule 2, again in cycle 3

Comp(Part, Subpt) :-
Assembly(Part, Part2, Qty),
Comp(Part2, Subpt).

21

Cycle 4

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Comp tuples by applying Rule 2 again in cycle 3

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube
trike	rim
trike	tube

Union, New, Comp tuples by applying Rule 2, again in cycle 4

Comp(Part, Subpt) :-
Assembly(Part, Part2, Qty),
Comp(Part2, Subpt).

22

In cycle 4 we finished computing the transitive closure

- Cycle 5 – no new rows → achieved fixed point.
- But, we have, on cycle 5, recomputed the entire contents of the EDB

Independent of Efficiency… Have we forgotten something?

Yes, we've only spoken to computing the transitive closure.

What if we don't need the transitive closure, but have a more specific query.

e.g. What are the parts of the wheel?

In SQL, actually kind of nice:

```
WITH RECURSIVE Comp(Part, Subpt) AS
(SELECT A1.Part, A1.Subpt FROM Assembly A1)
UNION
(SELECT A2.Part, C1.Subpt
 FROM Assembly A2, Comp C1
 WHERE A2.Subpt=C1.Part)
```

General construction

```
SELECT * FROM Comp C2
   Where C2.part = wheel.
```

Specific query

The optimizer worries the rest

25

What are the optimizations?

- Avoid Repeated Inferences:
- Avoid Unnecessary Inferences:

26

What are the optimizations?

- Avoid Repeated Inferences:
  - Inference ≡ Query Evaluation
  - → Semi-naïve evaluation
- Avoid Unnecessary Inferences:
  - Magic-set transformation

27

Intuition From SQL Version

If the following were actually a materialized view:

```
WITH RECURSIVE Comp(Part, Subpt) AS
(SELECT A1.Part, A1.Subpt FROM Assembly A1)
UNION
(SELECT A2.Part, C1.Subpt
 FROM Assembly A2, Comp C1
 WHERE A2.Subpt=C1.Part)
```

Given query:

```
SELECT * FROM Comp C2
   WHERE C2.part = trike;
```

The answer is the entire transitive closure of trike

Computer everything – no waste

28

Avoiding Unnecessary Inferences

If the following were a materialized view:

```
WITH RECURSIVE Comp(Part, Subpt) AS
(SELECT A1.Part, A1.Subpt FROM Assembly A1)
UNION
(SELECT A2.Part, C1.Subpt
 FROM Assembly A2, Comp C1
 WHERE A2.Subpt=C1.Part)
```

Given query:

```
SELECT * FROM Comp C2
   WHERE C2.part = wheel.
```

From the entire materialized view,
Answer = {spoke, tire, rim, tube}

So we did not need to compute the entire transitive closure.

→ Much of initial query execution was not necessary to determine the answer.

29

Same explanation: strict Datalog terminology

- Unnecessary inferences:
- If we just want to find components of a particular part, say wheel, then first computing general fixpoint of Comp program and then at end selecting tuples with wheel in the first column is wasteful.
- This would compute many irrelevant facts.

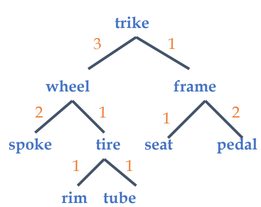
30

Magic Sets [**a different**] Example: Avoiding Unnecessary Inferences

```
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  Assembly(P1,S2,Q2).
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  SameLev(P1,P2),
                  Assembly(P2,S2,Q2).
```

Observe: Much like the cousin example

- Semantics?



31

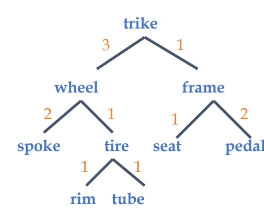
Example for considering Avoiding Unnecessary Inferences

```
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  Assembly(P1,S2,Q2).

SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  SameLev(P1,P2),
                  Assembly(P2,S2,Q2).
```

Observe: Much like the cousin example
Sibling(S1, S2):= Par(P1, S1,_),
Par(P1, S2,_)

- Semantics?

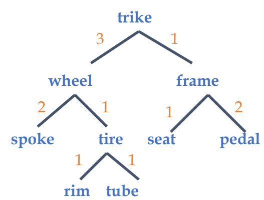


32

Avoiding Unnecessary Inferences

```
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  Assembly(P1,S2,Q2).
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
                  SameLev(P1,P2),
                  Assembly(P2,S2,Q2).
```

- Tuple (S1,S2) in SameLev if there is path up from S1 to some node and down to S2 with same number of up and down edges.



33

What if materialize the program
Apply Rule 1

SameLev₀(S1,S2) :-

Wheel	Frame
Spoke	Tire
Seat	Pedal
Rim	Tube

Assembly(P1,S1,Q1).

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

⋈

Assembly(P1,S2,Q2)

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

34

Apply Rule 2, first time

SameLev(S1,S2) :- Assembly(P1,S1,Q1), SameLev(P1,P2), Assembly(P2,S2,Q2).

Assembly(P1,S1,Q1)

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

SameLev₀(P1,P2),

Wheel	Frame
Spoke	Tire
Seat	Pedal
Rim	Tube

Column Break

Assembly(P2,S2,Q2)

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

35

Result

SameLev(S1,S2) :-

Rule 1

Rule 2, first application.. reaches fixed point

Wheel	Frame
Spoke	Tire
Seat	Pedal
Rim	Tube
Spoke	Seat
Spoke	Pedal
Tire	Seat
Tire	Pedal

36

Example Query

- Want all SameLev tuples with spoke in first column.

In Datalog,

SameLev(spoke,s):=.

	Wheel	Frame
✓	Spoke	Tire
	Seat	Pedal
	Rim	Tube
✓	Spoke	Seat
✓	Spoke	Pedal
	Tire	Seat
	Tire	Pedal

37

Approach:

- Recall, ordinary queries, pushing selects down (usually a constant) → big improvements
- We will augment a Datalog program sets of constants, *Magic Sets*
  - Defined by new predicates, that yield the *Magic Sets* of constants
  - Siubstute them into the Datalog program

	Wheel	Frame
✓	Spoke	Tire
	Seat	Pedal
	Rim	Tube
✓	Spoke	Seat
✓	Spoke	Pedal
	Tire	Seat
	Tire	Pedal

38

Avoiding Unnecessary Inferences

- Want all SameLev tuples with spoke in first column.
- Intuition: "Push" this selection into fixpoint computation.

SameLev(S1,S2) :-
Assembly(P1,S1,Q1),
SameLev(P1,P2),
Assembly(P2,S2,Q2).

SameLev(**spoke**,S2) :-
Assembly(P1,**spoke**,Q1),
SameLev(**P1?=spoke?**,P2),
Assembly(P2,S2,Q2).

39

Avoiding Unnecessary Inferences

- Intuition: "Push" this selection with spoke into fixpoint computation.
  - Substitute S1 with spoke .

SameLev(S1,S2) :-
Assembly(P1,S1,Q1),
SameLev(P1,P2),
Assembly(P2,S2,Q2).

SameLev(**spoke** ,S2) :-
Assembly(P1,**spoke**,Q1),
SameLev(**P1**,P2),
Assembly(P2,S2,Q2).

SameLev(**spoke**,seat) :- Assembly(wheel,**spoke**,2),
SameLev(wheel,frame), Assembly(frame,seat,1).

- Other SameLev tuples are needed to compute all such tuples with spoke,

40

"Magic Sets" Idea

- 1. Define "filter" table that computes all relevant values
- 2. Restrict computation of SameLev to infer only tuples with relevant value in first column.

41

Intuition

- Relevant values: contains all tuples m for which we require to compute all same-level tuples with m in first column to answer query.
- Put differently, relevant values are all Same-Level tuples whose first field contains value on path from spoke up to root.
- We call it Magic-SameLevel (Magic-SL)

42

"Magic Sets" in Example

- Idea: Define "filter" table that computes all relevant values : Collect all parents of spoke.

Magic_SL(P1) :- Magic_SL(S1), Assembly(P1,S1,Q1).
Magic_SL(spoke) :- .

Make Magic table as Magic-SL
Notice:

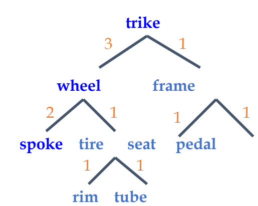
- rule with head, Magic_SL(p1), has no constants
- Magic_SL(spoke):-. Is a separate rule for defining the filter table

43

Define "filter" table that computes all relevant values :

- Idea: Collect all parents of spoke.

Magic_SL(P1) :- Magic_SL(S1), Assembly(P1,S1,Q1).
Magic_SL(spoke) :- .



Make Magic table as Magic-SameLevel.

44

"Magic Sets" Idea

- Idea: Define a "filter" table (in the form a predicate) to restrict the computation of SameLev to only tuples with a relevant value in the first column

Define:
Magic_SL(P1) :- Magic_SL(S1), Assembly(P1,S1,Q1).
Magic(spoke).

Add predicate to same level rules (limiting the results to a subset of the original
SameLev(S1,S2) :- **Magic_SL(S1),** Assembly(P1,S1,Q1),
Assembly(P1,S2,Q2).

SameLev(S1,S2) :- **Magic_SL(S1),** Assembly(P1,S1,Q1),
SameLev(P1,P2), Assembly(P2,S2,Q2).

45

The Magic Sets Algorithm

- 1. Generate an "adorned" program
  - Program is rewritten to make pattern of bound and free arguments in query explicit
- 2. Add magic filters of form "Magic_P"
  - for each rule in adorned program add a Magic condition to body that acts as filter on set of tuples generated (predicate P to restrict these rules)
- 3. Define new rules to define filter tables
  - Define new rules to define filter tables of form Magic_P

47

Step 1:Generating Adorned Rules

- Adorned program for query pattern SameLev^{bf}, *assuming right-to-left order of rule evaluation* :

SameLev^{bf} (S1,S2) :- Assembly(P1,S1,Q1), Assembly(P1,S2,Q2).

SameLev^{bf} (S1,S2) :- Assembly(P1,S1,Q1),

SameLev^{bf} (P1,P2), Assembly(P2,S2,Q2).

- Argument of (a given body occurrence of) SameLev is:
  - b if it appears to the left in body,
  - or if it is a b argument of head of rule,
  - Otherwise it is *free. (as in won't be bound)*
- Assembly not adorned because explicitly stored table (EDB).

48

Step 2: Add Magic Filters

- For every rule in adorned program add a 'magic filter' predicate

SameLev^{bf} (S1,S2) :- Magic_SL (S1), Assembly(P1,S1,Q1), Assembly(P1,S2,Q2).

SameLev^{bf} (S1,S2) :- Magic_SL (S1),
Assembly(P1,S1,Q1),
SameLev^{bf} (P1,P2), Assembly(P2,S2,Q2).

- Filter predicate: copy of head of rule, Magic prefix, and delete free variable

49

Step 3:Defining Magic Tables

- Rule for Magic_P is generated from each occurrence of recursive P in body of rule:
  - Delete everything to right of P
  - Add prefix “Magic” and delete free columns of P
  - Move P, with these changes, into head of rule

Step 3:Defining Magic Table

- Rule for Magic_P is generated from each occurrence O of recursive P in body of rule:
  - Delete everything to right of P

SameLev^{bf} (S1,S2) :- Magic_SL(S1), Assembly(P1,S1,Q1),
SameLev^{bf} (P1,P2), Assembly(P2,S2,Q2).

  - Add prefix “Magic” and delete free columns of P

Magic-SameLev^{bf} (S1,S2) :- Magic_SL(S1), Assembly(P1,S1,Q1),
Magic-SameLev^{bf} (P1 __).

  - Move P, with these changes, into head of rule

Magic_SL(P1) :- Magic_SL(S1), Assembly(P1,S1,Q1).

Supplemental Slides:
Semi-naïve Evaluation Example

Beginning Cycle 1

Comp

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Comp, is an empty EDB table

Rule 2 is not satisfied

Eval rule 1, and get

Rule 1: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

End Cycle 1

Comp (1)

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Rule 1: Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

56

Cycle 2

Assembly

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Comp (1)

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Rule 1, no new assembly rows, do nothing, but Comp 1 stays:

trike	wheel
trike	frame
frame	seat
frame	pedal
wheel	spoke
wheel	tire
tire	rim
tire	tube

Rule 2: Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).

Compute Comp tuples by applying Rule 2 (in cycle 2)

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

57

Cycle 3

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Consider only these, the new rows of Comp

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube
trike	rim
trike	tube

Per incremental view maintenance, only these two rows get added

Comp(Part, Subpt) :-
Assembly(Part, Part2, Qty),
Comp(Part2, Subpt).

58

Cycle 4

trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

Assembly instance

Join new rows only.
Join result is empty.
No more work to do
→ fixed point,
No cycle 5

trike	rim
trike	tube

Comp(Part, Subpt) :-
Assembly(Part, Part2, Qty),
Comp(Part2, Subpt).

59