Ensuring Data Integrity Without Deferred Constraint Checking

Efficient Multiple Simultaneous Assignment of Relvars

By Adrian Hudnott, Hugh Darwen & Jane Sinclair

THE UNIVERSITY OF WARWICK

Multiple Simultaneous Assignment

Why?

1. Constraints that make sequential updates impossible.
2. Mutually dependent updates.
3. Performance.

Why Not Deferred Constraint Checking?

1. Transaction sees inconsistent database.
2. Deferred / immediate depends on DB design.
3. Semantic optimization using constraints.
4. User-defined operators loose orthogonality.

Mutually Dependent Updates

- Credit reference agency.
- 2 entities: credit history entries and credit reports.
- Creditor performs a credit check.
- Produces a credit report based on *previous* credit history.
- Creates a new history entry.
- History entry applies a penalty for a failed credit application.
- Amount of penalty depends on contents of *previous* credit reports.
- Credit report depends on *previous* credit history *and* credit history depends on outcome of *previous* reports.

Syntax

- V1 := Q1,… Vn := Qn;
- V1… Vn not necessarily unique.
- Syntactic sugar:
  - E.g. INSERT V Q is equivalent to V:= V UNION Q.

- Hugh Darwen, and C.J. Date, *Databases, Types, and the Relational Model: The Third Manifesto*. 3rd Ed. (USA: Pearson, 2006)

Naive Implementation

- Two sets of variables:
  - Program variables (e.g. `V`).
  - Temporaries (e.g. `V'`) (not used in the original program).
- Implement V1 := E1, V2 := E2, V3 := E3 as:

```
V1' := E1;  V2' := E2;  V3' := E3;
V1  := V1'; V2  := V2'; V3  := V3;
```

- Not efficient if `V1'` and `V1`, etc. have similar values.

Formal Semantics

$$\langle \mathtt{V} := \mathtt{E}, ACL \rangle, s \Rightarrow \langle replace(ACL, V, E) \rangle, s[V' \mapsto \mathrm{Exp}[\mathtt{E}]s]$$

$$\langle \mathtt{V} := \mathtt{E} \rangle, s \Rightarrow msa(s[V' \mapsto \mathrm{Exp}[\mathtt{E}]s])$$

$$msa(s) = \{V \mapsto x \mid declared(V, s) \wedge assign(V, s) = x\}$$

$$assign(V, s) = \begin{cases} s(V') & \text{if } V' \in \mathrm{dom}(s) \\ s(V) & \text{otherwise} \end{cases}$$

$$replace(, VR, ER) =$$

$$replace(\mathtt{V} := \mathtt{E}, ACL, VR, ER) = \begin{cases} \mathtt{V} := \mathtt{E}[ER/VR], ACL & \text{if } V = VR \\ \mathtt{V} := \mathtt{E}, replace(ACL, VR, ER) & \text{if } V <> VR \end{cases}$$

Related Problems

- Parallel Programming: Executing a program for Concurrent Read Exclusive Write (CREW) synchronous PRAM on a Von Neumann machine.
- Transaction Serialization: Finding a conflict-free schedule (but no guarantee that one exists!).

Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. Merge Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation

---

10. Execution

Optimization

1. Expansion
2. **Conversion to Canonical Form**
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. Merge Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation
10. Execution

Canonical Form

- Already seen an example:
  - INSERT V Q $\rightarrow$ V := UNION{V, Q}
- Also, trivial things:
  - UNION{Q} $\rightarrow$ Q
- Expansion and conversion to canonical form are *abstractions*.
  - In practice the work is divided between the existing parsing and query rewrite steps.

Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. **Dependency Graph Construction**
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. Merging Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation
10. Execution

Updates

- Two types of updates: “in place” and not “in place”.
- “In place” updates are implemented using either overwriting or delta compression.
- Not “in place” updates are implemented using shadow copying.

Dependency Graphs

- A query *Q* is *independent of the update V:=E* iff Q ≡ Q[E/V].
- For queries Q1 and Q2, *Q1 contains Q2* iff Q1 ⊇ Q2.
- A labelled directed graph *G* = (*V*, *E,* cost) is a dependency graph for a multiple assignment V1:=Q1,… Vn:=Qn iff:

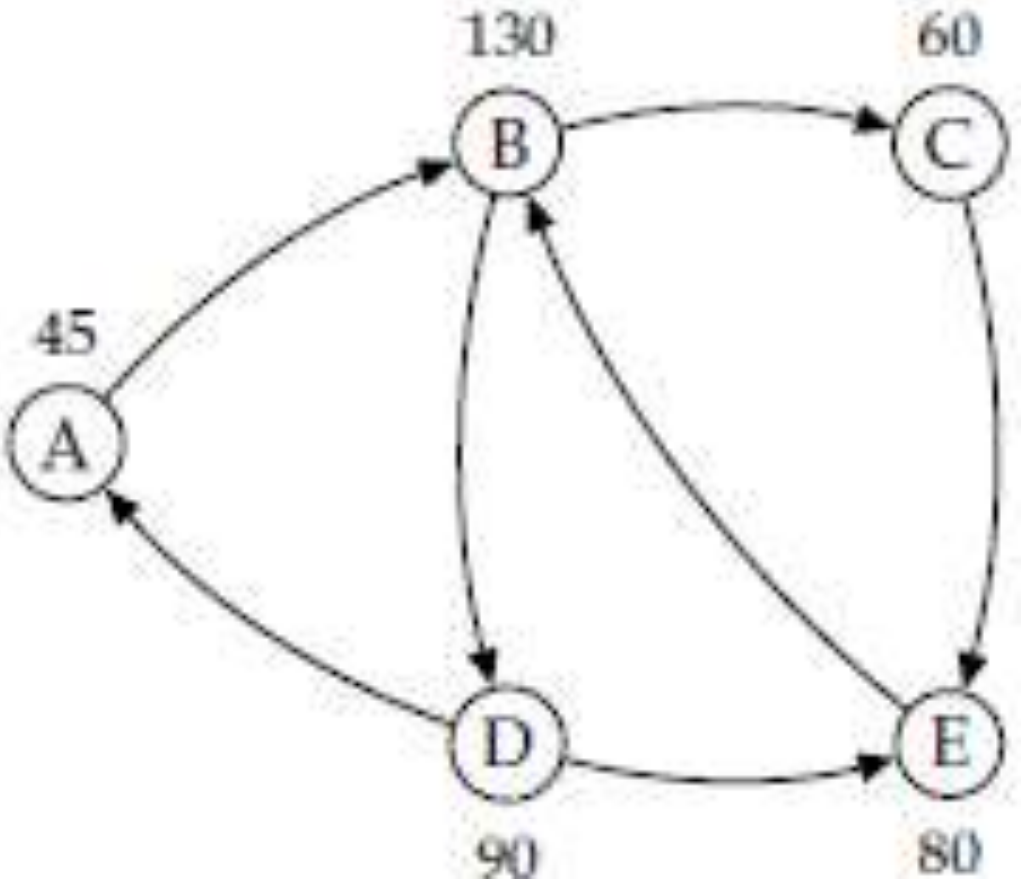
$$V = \{\mathrm{V1}, \ldots \mathrm{Vn}\}$$

$$E = \{v \rightarrow w \mid assigned(v, q) \wedge assigned(w, r) \wedge \neg(q \equiv q[r / w])$$

$$\mathrm{cost}(v) = \min(\mathrm{inplace}(v), \mathrm{notinplace}(v))$$

- *G* can contain loops.
- $v \rightarrow w$ means "*v* must be done before *w*".

Dependency Graphs



Independence Checking

- Need to show Qi $\subseteq$ Qi[Qj/Vj] and Qi[Qj/Vj] $\subseteq$ Qi (subject to integrity constraints).
- Carles Farre, Ernest Teniente, and Toni Urpi. (2005). Checking query containment with the CQC method. *Data and Knowledge Engineering, 53*(2), 163-223.
  - An automated method of performing proofs for query containment of Datalog queries with negation.
  - Tries to find a minimal database that violates containment.
  - Uses a tableau with 5 columns.

Farre et al. DATAK 53(2) 2005

- Search for a counterexample – a tuple in Qi MINUS Qi[Qj/Vi].

a b c d e

Goal	Conditions to Enforce	Example Database	Conditions to Maintain	Literals Used

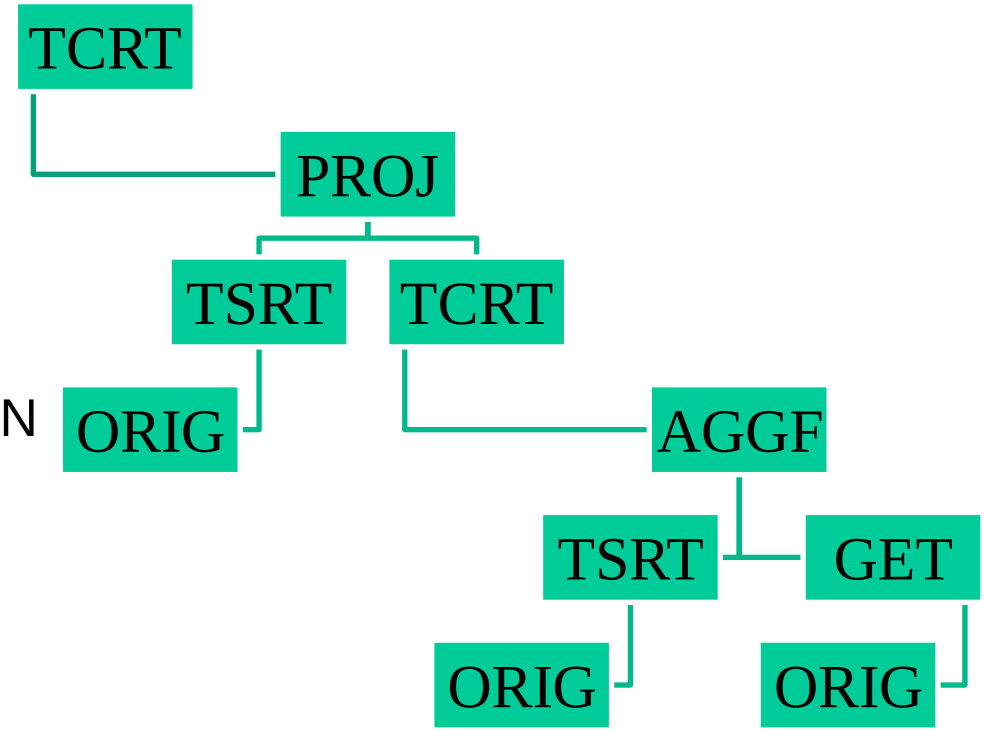
- Expansion.
- Make database satisfy non-negated query clause ($a \rightarrow c,e$).
- Make negated query clause into a constraint ($a \rightarrow b$).
- Check a constraint ($b \rightarrow d,e$).
- Recheck a constraint ($d \rightarrow b$).

Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. **Multi-Query Optimization**
6. Cyclic Dependency Removal
7. Merge Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation
10. Execution

Plan Enumeration & Selection (Single Query)

```
SELECT
DISTINCT "S#"
FROM S
WHERE STATUS * 100 <
(SELECT N FROM
   (SELECT "S#", COUNT(*) AS N
      FROM SP
      GROUP BY "S#") AS SPA
   WHERE "S.S#" = "SPA.S#")
```



Multi-Query Optimization

- Identifies two or more equivalent subexpressions within one or more queries.
- Cost saving: (n-1) x evaluation cost.
- Costs *incurred*:
  - 1 x evaluation cost remains.
  - 1 x spooling result out.
  - n x reading result in.
- Problem is no longer compositional.
- Zhou, Freytag, Larson, and Lehner, 'Efficient Exploitation of Similar Subexpressions for Query Processing', *SIGMOD ’07.*

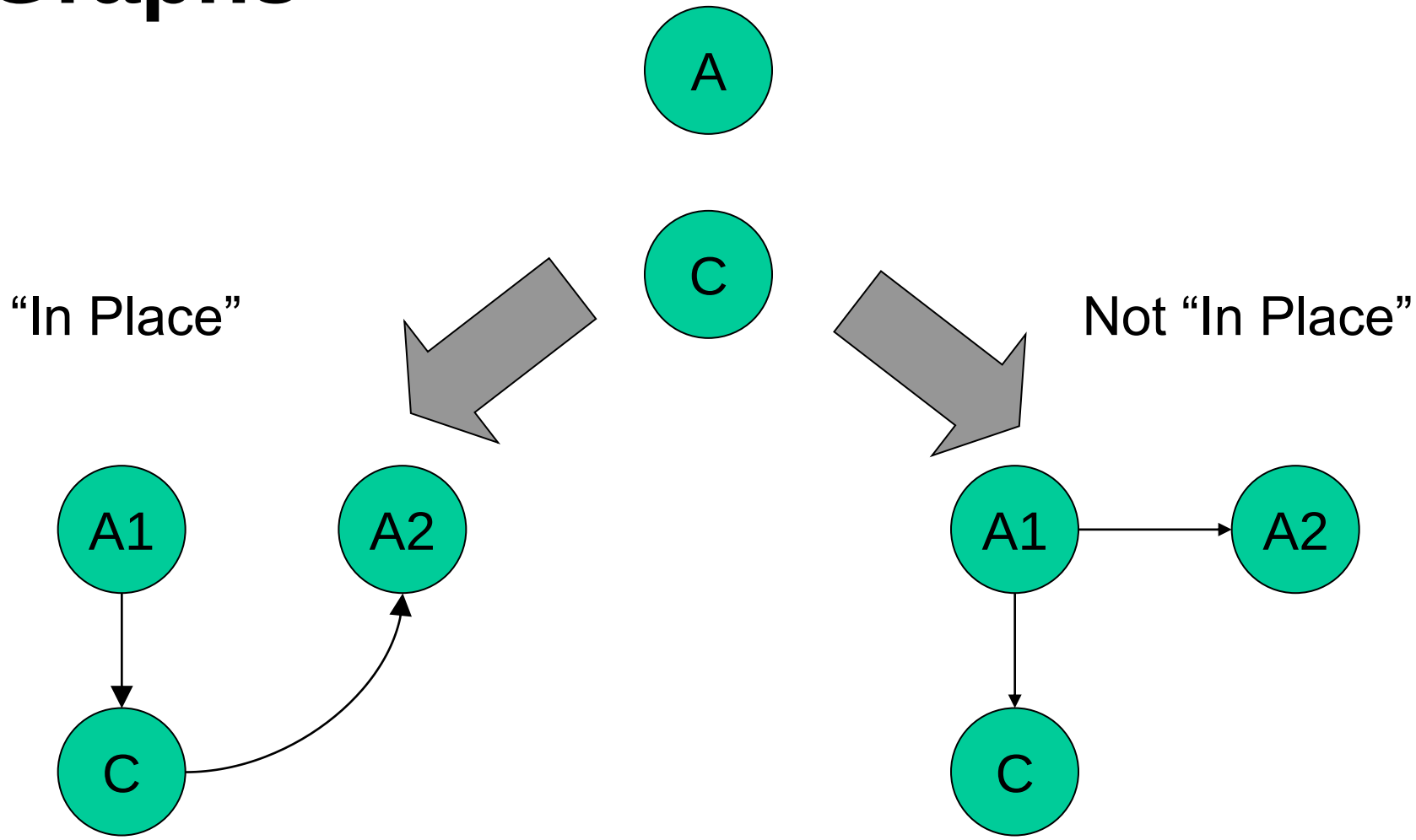
Approaches to MQO

- Find matching expressions by:
  - Textual match
  - Search parse tree
  - Graph representation
  - Most beneficial view
- Costing:
  - Delayed
  - Heuristic (best plans without MQO, greedy, A*, GA)

MQO (Zhou et al.)

1. Note the source relvars for each relational expression.
2. Generate candidate common subexpressions (CSEs).
   - Join compatible (with renaming).
   - Covering.
   - Merge similar CSEs.
   - Heuristics (no cheap CSEs, no CSEs with huge results, no large child CSEs).
3. Optimize with CSEs.
   - Delayed costing.
   - Optimize with different sets of CSEs enabled.

MQO & Extended Dependency Graphs



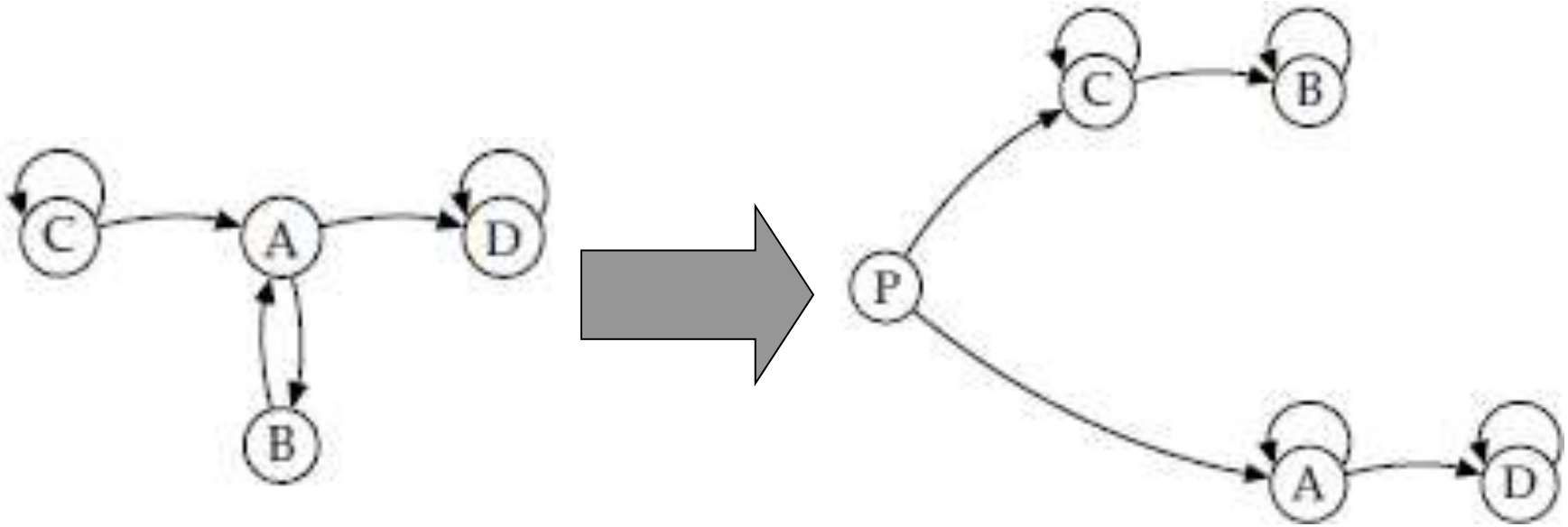
Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. **Cyclic Dependency Removal**
7. Merge Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation
10. Execution

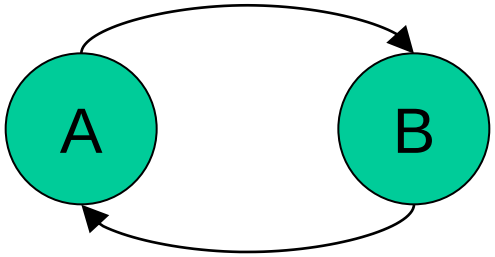
Cyclic Dependency Removal

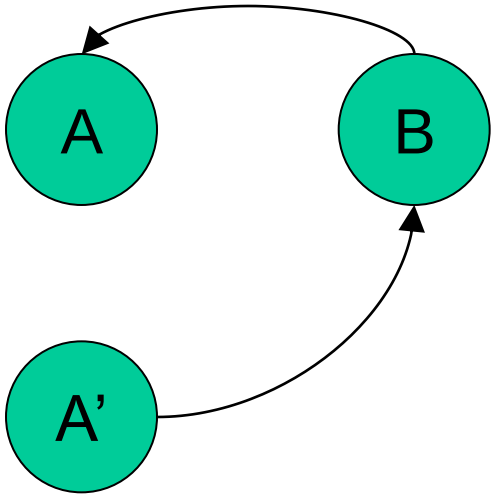
- 4 Methods:
  - Permutation
  - Vertex Splitting
  - Multi-Query Optimization
  - Multi-Query De-optimization

Permutation Method



Vertex Split Method





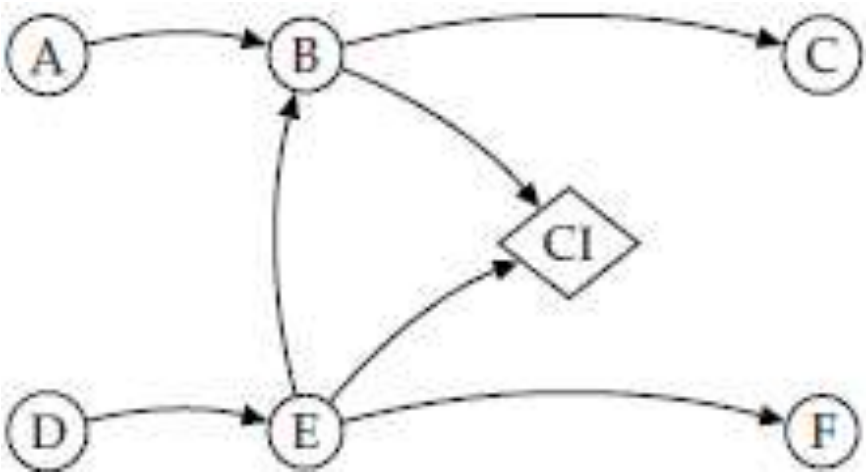
Minimum Feedback Vertex Set

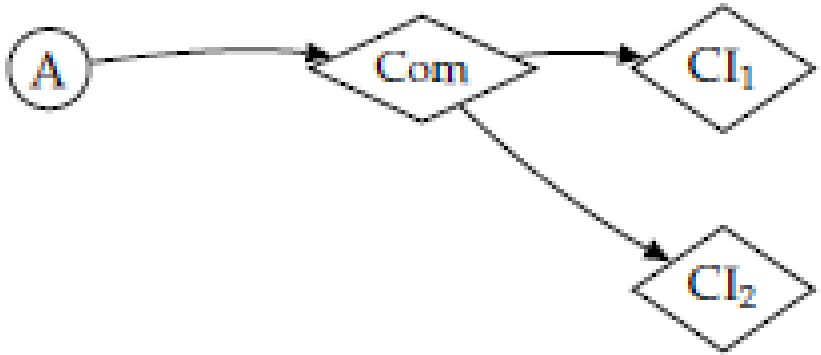
- A *feedback vertex set* of a directed graph is a subset of its vertices containing at least one vertex for each directed cycle.
- The minimum feedback vertex set problem consists of finding a smallest cost feedback vertex set.
- Splitting a vertex that lies on many cycles could be advantageous.
- But not necessarily if the vertex has a high cost.
- Demetrescu, and Finocchi, 'Combinatorial Algorithms for Feedback Problems in Directed Graphs', *Information Processing Letters,* 86(3).

Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. **Merge Integrity Constraint Checks**
8. Transitive Dependency Removal
9. Code Generation
10. Execution

Merge Integrity Constraints





Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. Merge Integrity Constraint Checks
8. **Transitive Dependency Removal**
9. Code Generation
10. Execution

Transitive Dependencies

- Clean up the dependency graph.
  - Important for a distributed DBMS.
- Uses Boolean matrix algebra.

$$E \wedge \neg\left(E^2 \cdot E^*\right)$$

- Same complexity as matrix multiplication.
- Kozen, *The Design and Analysis of Algorithms*, *Texts and Monographs in Computer Science* (New York: Springer, 1992), p.27,240.

Optimization

1. Expansion
2. Conversion to Canonical Form
3. Query Rewrite
4. Dependency Graph Construction
5. Multi-Query Optimization
6. Cyclic Dependency Removal
7. Merge Integrity Constraint Checks
8. Transitive Dependency Removal
9. Code Generation
10. **Execution**

Execution

- Parallel execution
- Performing an assignment:
  - Like SQL MERGE with delete extension like MS SQL Server.
  - Implicit deletion
  - Delete-only assignments

Delta Compression

- Compact representation of old and new value.
- Used in DBMSs (Oracle, MS SQL Server, Firebird) for *multi-version concurrency control* (MVCC).
- Various methods, research papers…

Summary

- Multiple simultaneous assignment is a better alternative to deferred constraint checking.
- Challenging to implement. System must:
  - Identify dependencies (dependency graphs, query independent of update).
  - Exploit dependencies (MQO).
  - Handle mutual dependencies (permutation, delta compression, shadow copying).
  - Implement assignment!
  - Schedule parallel code.