A SURVEY OF INCREMENTAL REASONING ALGORITHMS FOR DATALOG VARIANTS

Boris Motik

University of Oxford

December 8, 2016
TABLE OF CONTENTS

1 Incremental Reasoning in Datalog

2 Counting-Based Algorithms

3 The Delete/Rederive (DRed) Algorithm

4 The Forward/Backward/Forward (FBF) Algorithm

5 Comparing the Three Algorithms

6 Extending Datalog: Stratified Negation and Equality

7 Other Approaches: FOIESs & Truth Maintenance Systems

8 Conclusion
# Table of Contents

1. Incremental Reasoning in Datalog

2. Counting-Based Algorithms

3. The Delete/Rederive (DRed) Algorithm

4. The Forward/Backward/Forward (FBF) Algorithm

5. Comparing the Three Algorithms

6. Extending Datalog: Stratified Negation and Equality

7. Other Approaches: FOIESs & Truth Maintenance Systems

8. Conclusion
Incremental Reasoning in Datalog

**Datalog and its Application**

- **Datalog**: a DB query language / KR formalism based on if-then rules

**Example Datalog Program**

\[
\text{hasParent}(x, y) \leftarrow \text{hasMother}(x, y) \\
\text{hasAunt}(x, z) \leftarrow \text{hasParent}(x, y) \land \text{hasSister}(x, y)
\]

- **Applications of datalog**:
  - **Semantic Web reasoning**
    - RDFS and OWL 2 RL inference rules can be encoded into datalog
    - Supported in graphDB, Systap Big Data, Oracle DB, MarkLogic, RDFox, . . .
  - **Data integration**
    - Datalog rules (possibly with function symbols) can express source mappings
  - **Enterprise data management**
    - Datalog rules capture data dependencies and thus simplify maintenance
    - Supported in LogicBlox
DATALOG REASONING VIA MATERIALISATION

- Main task: compute entailments of program and a set of facts

**EXAMPLE DATALOG PROGRAM**

\[
\text{hasParent}(x, y) \leftarrow \text{hasMother}(x, y) \\
\text{hasAunt}(x, z) \leftarrow \text{hasParent}(x, y) \land \text{hasSister}(x, y)
\]

**EXAMPLE SET OF FACTS**

\[
\text{hasMother}(Peter, Paula) \\
\text{hasSister}(Paula, Ann)
\]

**EXAMPLE ENTAILMENTS**

\[
\text{hasParent}(Peter, Paula) \\
\text{hasAunt}(Peter, Ann)
\]

- Materialisation: apply rules, store consequences, repeat while change
- Main benefit: all consequences are precomputed during preprocessing
- Main drawback: dealing with changes in the input
Incremental Reasoning in Datalog

INCREMENTAL REASONING

Inputs:
- a set of explicit facts $E$
- a datalog program $P$
- an ‘old’ materialisation $I$ of $P$ on $E$
- a set $E^-$ of facts to delete from $E$ and a set $E^+$ of facts to add to $E$

Goal: compute the ‘new’ materialisation of $P$ on $(E \setminus E^-) \cup E^+$
**Inputs:**
- a set of *explicit* facts $E$
- a datalog program $P$
- an ‘old’ materialisation $I$ of $P$ on $E$
- a set $E^-$ of facts to delete from $E$ and a set $E^+$ of facts to add to $E$

**Goal:** compute the ‘new’ materialisation of $P$ on $\left( E \setminus E^- \right) \cup E^+$
- **Do this efficiently** $\Rightarrow$ without recomputing most of $I$
**Incremental Reasoning**

- **Inputs:**
  - a set of *explicit* facts $E$
  - a datalog program $P$
  - an ‘old’ materialisation $I$ of $P$ on $E$
  - a set $E^-$ of facts to delete from $E$ and a set $E^+$ of facts to add to $E$

- **Goal:** compute the ‘new’ materialisation of $P$ on $(E \setminus E^-) \cup E^+$
  - Do this efficiently $\Rightarrow$ without recomputing most of $I$

- **Note:** we delete only *explicit* facts
  - Users cannot delete inferred facts
  - $\Rightarrow$ That is a subject for *belief revision*
Key Issues

- Addition usually not a problem: just use the seminaïve algorithm
- Deleting facts with multiple derivations is a source of difficulty

Example Program

\[ A(y) \leftarrow A(x) \land R(x, y) \]

Example Data & Consequences

<table>
<thead>
<tr>
<th>Explicit Data</th>
<th>Consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(a) ), ( R(a, c) ), ( R(a, d) ), ( A(b) ), ( R(b, c) )</td>
<td>( A(c) ), ( A(d) )</td>
</tr>
</tbody>
</table>

- Delete \( A(a) \)
Key Issues

- Addition usually not a problem: just use the seminaïve algorithm
- Deleting facts with multiple derivations is a source of difficulty

Example Program

\[ A(y) \leftarrow A(x) \land R(x, y) \]

Example Data & Consequences

<table>
<thead>
<tr>
<th>Explicit Data</th>
<th>Consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(a), R(a, c), R(a, d), A(b), R(b, c) )</td>
<td>( A(c), A(d) )</td>
</tr>
</tbody>
</table>

- Delete \( A(a) \)
- \( A(c) \) ‘survives’
  - \( A(b) \) and \( R(b, c) \) still derive \( A(c) \)
**Key Issues**

- Addition usually not a problem: just use the seminaïve algorithm
- Deleting facts with multiple derivations is a source of difficulty

**Example Program**

\[ A(y) \leftarrow A(x) \land R(x, y) \]

**Example Data & Consequences**

<table>
<thead>
<tr>
<th>Explicit Data</th>
<th>Consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(a), R(a, c), R(a, d), A(b), R(b, c) )</td>
<td>( A(c), A(d) )</td>
</tr>
</tbody>
</table>

- Delete \( A(a) \)
- \( A(c) \) ‘survives’
  - \( A(b) \) and \( R(b, c) \) still derive \( A(c) \)
- \( A(d) \) ‘dies’
  - There is no alternative derivation of \( A(d) \)
Rematerialisation (i.e., restarting from scratch) is a possible solution

Ideally, incremental reasoning should be faster than rematerialisation
- Unrealistic requirement: if $E^- = E$, rematerialisation requires no work
- $\Rightarrow$ An incremental algorithm will necessarily do more work

Next best goal for worst-case performance:
- Undo precisely rule instances that do not hold any more, and apply precisely new rule instances
- $\Rightarrow$ Still not attainable in most cases

Weak worst-case performance:
- Do not perform more work than in the ‘old’ and the ‘new’ materialisation combined

Overall goal: more efficient than rematerialisation on small $E^-$ and $E^+$
# Table of Contents

1. Incremental Reasoning in Datalog

2. Counting-Based Algorithms

3. The Delete/Rederive (DRed) Algorithm

4. The Forward/Backward/Forward (FBF) Algorithm

5. Comparing the Three Algorithms

6. Extending Datalog: Stratified Negation and Equality

7. Other Approaches: FOIESs & Truth Maintenance Systems

8. Conclusion
Basic Counting (Nonrecursive Variant)

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \]

<table>
<thead>
<tr>
<th>A(a)</th>
<th>B(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Associate with each fact a counter initialised to zero. Increment the counter after each derivation. Delete \( A(a) \): Decrease its counter. The counter of \( A(a) \) reaches zero, so propagate deletion. Delete \( B(a) \): Decrease its counter. The counter of \( B(a) \) reaches zero, so propagate deletion.

Worst-case optimal for nonrecursive rules.
Basic Counting (Nonrecursive Variant)

**Example**

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
Basic Counting (Nonrecursive Variant)

**Example**

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation

<table>
<thead>
<tr>
<th>A(a)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(a)</td>
<td>1</td>
</tr>
<tr>
<td>C_0(a)</td>
<td>2</td>
</tr>
<tr>
<td>C_1(a)</td>
<td>1</td>
</tr>
</tbody>
</table>
Basic Counting (Nonrecursive Variant)

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(a))</td>
<td>1</td>
</tr>
<tr>
<td>(B(a))</td>
<td>1</td>
</tr>
<tr>
<td>(C_0(a))</td>
<td>2</td>
</tr>
<tr>
<td>(C_1(a))</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Basic Counting (Nonrecursive Variant)

Example

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>...</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_0(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_1(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_n(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic Counting (Nonrecursive Variant)

Example

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter

\[
\begin{array}{|c|c|}
\hline
A(a) & 1 \\
B(a) & 1 \\
C_0(a) & 2 \\
C_1(a) & 1 \\
\ldots & \\
C_n(a) & 1 \\
\hline
\end{array}
\]
**Basic Counting (Nonrecursive Variant)**

**Example**

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion

<table>
<thead>
<tr>
<th>( A(a) )</th>
<th>( B(a) )</th>
<th>( C_0(a) )</th>
<th>( C_1(a) )</th>
<th>...</th>
<th>( C_n(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>
Basic Counting (Nonrecursive Variant)

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion

| \( A(a) \) | 0 |
| \( B(a) \) | 1 |
| \( C_0(a) \) | 1 |
| \( C_1(a) \) | 1 |
| \ldots | 1 |
| \( C_n(a) \) | 1 |
Basic Counting (Nonrecursive Variant)

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion
- Delete \( B(a) \):
  - Decrease its counter
**Basic Counting (Nonrecursive Variant)**

### Example

\[
\begin{align*}
C_0(x) & \leftarrow A(x) \\
C_0(x) & \leftarrow B(x) \\
C_i(x) & \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\end{align*}
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion
- Delete \( B(a) \):
  - Decrease its counter
  - The counter of \( B(a) \) reaches zero, so propagate deletion
Basic Counting (Nonrecursive Variant)

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion
- Delete \( B(a) \):
  - Decrease its counter
  - The counter of \( B(a) \) reaches zero, so propagate deletion
### Basic Counting (Nonrecursive Variant)

#### Example

<table>
<thead>
<tr>
<th></th>
<th>( A(a) )</th>
<th>( B(a) )</th>
<th>( C_0(a) )</th>
<th>( C_1(a) )</th>
<th>...</th>
<th>( C_n(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0(x) ) ← ( A(x) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( C_0(x) ) ← ( B(x) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_i(x) ) ← ( C_{i-1}(x) ) for ( 1 \leq i \leq n )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion
- Delete \( B(a) \):
  - Decrease its counter
  - The counter of \( B(a) \) reaches zero, so propagate deletion

Worst-case optimal for nonrecursive rules

- **Boris Motik**

A Survey of Incremental Reasoning 6/20
# Basic Counting (Nonrecursive Variant)

| A(a) | 0 |
| B(a) | 0 |
| C_0(a) | 0 |
| C_1(a) | 0 |
| ... |   |
| C_n(a) | 0 |

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete $A(a)$:
  - Decrease its counter
  - The counter of $A(a)$ reaches zero, so propagate deletion
- Delete $B(a)$:
  - Decrease its counter
  - The counter of $B(a)$ reaches zero, so propagate deletion

$$C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n$$
Basic Counting (Nonrecursive Variant)

Example

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n
\]

- Associate with each fact a counter initialised to zero
- Increment the counter after each derivation
- Delete \( A(a) \):
  - Decrease its counter
  - The counter of \( A(a) \) reaches zero, so propagate deletion
- Delete \( B(a) \):
  - Decrease its counter
  - The counter of \( B(a) \) reaches zero, so propagate deletion
- **Worst-case optimal** for nonrecursive rules
PROBLEMS OF COUNTING AND RECURSION

**Example**

\[
B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x)
\]

| \(A(a)\)   | 1 |
| \(B(a)\)   | 1 |
PROBLEMS OF COUNTING AND RECURSION

**Example**

\[ B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(a) )</td>
<td>1</td>
</tr>
<tr>
<td>( B(a) )</td>
<td>2</td>
</tr>
</tbody>
</table>
Problems of Counting and Recursion

Example

\[ B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A(a)</td>
<td>2</td>
</tr>
<tr>
<td>B(a)</td>
<td>2</td>
</tr>
</tbody>
</table>
PROBLEMS OF COUNTING AND RECURSION

Example

\[ B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(a) )</td>
<td>2</td>
</tr>
<tr>
<td>( B(a) )</td>
<td>2</td>
</tr>
</tbody>
</table>
PROBLEMS OF COUNTING AND RECURSION

EXAMPLE

\[ B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x) \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(a)</td>
<td></td>
</tr>
<tr>
<td>B(a)</td>
<td>2</td>
</tr>
</tbody>
</table>
PROBLEMS OF COUNTING AND RECURSION

Example

\[ \begin{align*}
B(x) & \leftarrow A(x), \quad A(x) \leftarrow B(x) \\
A(a) & \quad 1 \\
B(a) & \quad 2
\end{align*} \]
Problems of Counting and Recursion

Example

$$B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x)$$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(a)$</td>
<td>1</td>
</tr>
<tr>
<td>$B(a)$</td>
<td>1</td>
</tr>
</tbody>
</table>
PROBLEMS OF COUNTING AND RECURSION

EXAMPLE

\[ B(x) \leftarrow A(x), \quad A(x) \leftarrow B(x) \]

<table>
<thead>
<tr>
<th>( A(a) )</th>
<th>( B(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Cyclic dependencies cause problems
- Similar to why garbage collection by reference counting does not
Counting-Based Algorithms

COUNTING AND RECURRENCE

**Example**

\[
\begin{align*}
C_0(x) & \leftarrow A(x) \\
C_0(x) & \leftarrow B(x) \\
C_i(x) & \leftarrow C_{i-1}(x) \quad \text{for } 1 \leq i \leq n \\
C_0(x) & \leftarrow C_n(x)
\end{align*}
\]

| \(A(a)\) | 1 |
| \(B(a)\) | 1 |
Counting-Based Algorithms

**Counting and Recursion**

### Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(a)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$B(a)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$C_0(a)$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration

$$C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \quad C_0(x) \leftarrow C_n(x)$$
**Counting-Based Algorithms**

**Counting and Recursion**

### Example

<table>
<thead>
<tr>
<th></th>
<th>$C_0(x)$</th>
<th>$C_0(x)$</th>
<th>$C_i(x)$</th>
<th>$C_0(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(a)$</td>
<td>$C(x)$</td>
<td>$B(x)$</td>
<td>$C(x)$</td>
<td>$C_0(x)$</td>
</tr>
<tr>
<td>$B(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_0(a)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration.
**Counting and Recursion**

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$C_0(x)$ ← $A(x)$</th>
<th>$C_0(x)$ ← $B(x)$</th>
<th>$C_i(x)$ ← $C_{i-1}(x)$ for $1 \leq i \leq n$</th>
<th>$C_0(x)$ ← $C_n(x)$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A(a)$</th>
<th>$B(a)$</th>
<th>$C_0(a)$</th>
<th>$C_1(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
### Example

| $A(a)$  | 1 |
| $B(a)$  | 1 |
| $C_0(a)$ | 2 |
| $C_1(a)$ | 1 |
| ...     |   |

- Associate with each fact an array of counters, one per iteration
Counting-Based Algorithms

Counting and Recursion

**Example**

- $C_0(x) \leftarrow A(x)$
- $C_0(x) \leftarrow B(x)$
- $C_i(x) \leftarrow C_{i-1}(x)$ for $1 \leq i \leq n$
- $C_0(x) \leftarrow C_n(x)$

<table>
<thead>
<tr>
<th>$A(a)$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B(a)$</td>
<td>1</td>
</tr>
<tr>
<td>$C_0(a)$</td>
<td>2</td>
</tr>
<tr>
<td>$C_1(a)$</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$C_n(a)$</td>
<td>1</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
Counting-Based Algorithms

**Counting and Recursion**

**Example**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(a)$</td>
<td>1</td>
<td>$C_0(a)$</td>
<td>2</td>
</tr>
<tr>
<td>$B(a)$</td>
<td>1</td>
<td>$C_1(a)$</td>
<td>1</td>
</tr>
<tr>
<td>$C_n(a)$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration

$C_0(x) \leftarrow A(x)$  
$C_0(x) \leftarrow B(x)$  
$C_i(x) \leftarrow C_{i-1}(x)$ for $1 \leq i \leq n$  
$C_0(x) \leftarrow C_n(x)$
Counting and Recursion

Example

\[ C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \quad C_0(x) \leftarrow C_n(x) \]

| \( A(a) \) | 0 |
| \( B(a) \) | 0 |
| \( C_0(a) \) | 2 1 |
| \( C_1(a) \) | 1 |
| \ldots \ |
| \( C_n(a) \) | 1 |

- Associate with each fact an array of counters, one per iteration
- Delete \( A(a) \) and \( B(a) \) by undoing derivations
COUNTING AND RECURSION

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$C_0(x)$ $\leftarrow A(x)$</th>
<th>$C_0(x)$ $\leftarrow B(x)$</th>
<th>$C_i(x)$ $\leftarrow C_{i-1}(x)$ for $1 \leq i \leq n$</th>
<th>$C_0(x)$ $\leftarrow C_n(x)$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A(a)$</th>
<th>$B(a)$</th>
<th>$C_0(a)$</th>
<th>$C_1(a)$</th>
<th>$\ldots$</th>
<th>$C_n(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
- Delete $A(a)$ and $B(a)$ by undoing derivations
### Example

<table>
<thead>
<tr>
<th>$A(a)$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B(a)$</td>
<td>0</td>
</tr>
<tr>
<td>$C_0(a)$</td>
<td>0</td>
</tr>
<tr>
<td>$C_1(a)$</td>
<td>0</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
</tr>
<tr>
<td>$C_n(a)$</td>
<td>1</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
- Delete $A(a)$ and $B(a)$ by undoing derivations
COUNTING AND RECURSION

**EXAMPLE**

<table>
<thead>
<tr>
<th></th>
<th>$A(a)$</th>
<th>$B(a)$</th>
<th>$C_0(a)$</th>
<th>$C_1(a)$</th>
<th>$C_n(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
- Delete $A(a)$ and $B(a)$ by undoing derivations
Counting-Based Algorithms

Counting and Recursion

**Example**

\[
\begin{align*}
C_0(x) & \leftarrow A(x) \\
C_0(x) & \leftarrow B(x) \\
C_i(x) & \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \\
C_0(x) & \leftarrow C_n(x)
\end{align*}
\]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(a))</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B(a))</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C_0(a))</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(C_1(a))</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C_n(a))</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Associate with each fact an array of counters, one per iteration
- Delete \(A(a)\) and \(B(a)\) by undoing derivations
Table of Contents

1 Incremental Reasoning in Datalog
2 Counting-Based Algorithms
3 The Delete/Rederive (DRed) Algorithm
4 The Forward/Backward/Forward (FBF) Algorithm
5 Comparing the Three Algorithms
6 Extending Datalog: Stratified Negation and Equality
7 Other Approaches: FOIESs & Truth Maintenance Systems
8 Conclusion
Example

\[
\begin{align*}
C_0(x) &\leftarrow A(x) \\
C_0(x) &\leftarrow B(x) \\
C_i(x) &\leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \\
C_0(x) &\leftarrow C_n(x)
\end{align*}
\]

Materialise initial facts
Delete \( A(a) \) using DRed:

1. Delete all facts with a derivation from \( A(a) \)
2. Rederive facts that have an alternative derivation

\[
\begin{align*}
A(a) \\
B(a)
\end{align*}
\]
Delete/Rederive (DRed) Algorithm at a Glance

Delete/Rederive (DRed): state of the art incremental maintenance algorithm

**Example**

| $C_0(x) \leftarrow A(x)$ | $C_0(x) \leftarrow B(x)$ | $C_i(x) \leftarrow C_{i-1}(x)$ for $1 \leq i \leq n$ | $C_0(x) \leftarrow C_n(x)$ |

- Materialise initial facts

$$
\begin{align*}
A(a) \\
B(a) \\
C_0(a) \\
C_1(a) \\
\ldots \\
C_n(a)
\end{align*}
$$
The DRed Algorithm at a Glance

Delete/Rederive (DRed): state of the art incremental maintenance algorithm

**Example**

\[
C_0(x) \leftarrow A(x) \quad C_0(x) \leftarrow B(x) \quad C_i(x) \leftarrow C_{i-1}(x) \text{ for } 1 \leq i \leq n \quad C_0(x) \leftarrow C_n(x)
\]

- Materialise initial facts
- Delete \( A(a) \) using DRed:

\[
\begin{array}{l}
A(a) \\
B(a) \\
C_0(a) \\
C_1(a) \\
\ldots \\
C_n(a)
\end{array}
\]
The DRed Algorithm at a Glance

Delete/Rederive (DRed): state of the art incremental maintenance algorithm

**Example**

| \(C_0(x) \leftarrow A(x)\) | \(C_0(x) \leftarrow B(x)\) | \(C_i(x) \leftarrow C_{i-1}(x)\) for \(1 \leq i \leq n\) | \(C_0(x) \leftarrow C_n(x)\) |

- Materialise initial facts
- Delete \(A(a)\) using DRed:
  1. Delete all facts with a derivation from \(A(a)\)

\[
\begin{align*}
A(a) \\
B(a) \\
C_0(a) \\
C_1(a) \\
\ldots \\
C_n(a)
\end{align*}
\]

\[
\begin{align*}
C_0^D(x) & \leftarrow A^D(x) \\
C_0^D(x) & \leftarrow B^D(x) \\
C_i^D(x) & \leftarrow C_{i-1}^D(x)\text{ for }1 \leq i \leq n \\
C_0^D(x) & \leftarrow C_n^D(x)
\end{align*}
\]
The DRed Algorithm at a Glance

Delete/Rederive (DRed): state of the art incremental maintenance algorithm

**Example**

| \( C_0(x) \leftarrow A(x) \) | \( C_0(x) \leftarrow B(x) \) | \( C_i(x) \leftarrow C_{i-1}(x) \) for \( 1 \leq i \leq n \) | \( C_0(x) \leftarrow C_n(x) \) |

- Materialise initial facts
- Delete \( A(a) \) using DRed:
  1. Delete all facts with a derivation from \( A(a) \)
     
     \[
     \begin{align*}
     C_0^D(x) & \leftarrow A^D(x) \\
     C_0^D(x) & \leftarrow B^D(x) \\
     C_i^D(x) & \leftarrow C_{i-1}^D(x) \text{ for } 1 \leq i \leq n \\
     C_0^D(x) & \leftarrow C_n^D(x)
     \end{align*}
     \]
  2. Rederive facts that have an alternative derivation
     
     \[
     \begin{align*}
     C_0(x) & \leftarrow C_0^D(x) \land A(x) \\
     C_0(x) & \leftarrow C_0^D(x) \land B(x) \\
     C_i(x) & \leftarrow C_i^D(x) \land C_{i-1}(x) \text{ for } 1 \leq i \leq n \\
     C_0(x) & \leftarrow C_0^D(x) \land C_n(x)
     \end{align*}
     \]
# Table of Contents

1. Incremental Reasoning in Datalog

2. Counting-Based Algorithms

3. The Delete/Rederive (DRed) Algorithm

4. The Forward/Backward/Forward (FBF) Algorithm

5. Comparing the Three Algorithms

6. Extending Datalog: Stratified Negation and Equality

7. Other Approaches: FOIESs & Truth Maintenance Systems

8. Conclusion
Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

\[
\begin{align*}
A(a) \\
B(a) \\
C_0(a) \\
C_1(a) \\
\ldots \\
C_n(a)
\end{align*}
\]

Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete \( A(a) \) using FBF:

\[
\begin{align*}
A(a) & \\
B(a) & \\
C_0(a) & \\
C_1(a) & \\
\vdots & \\
C_n(a) & 
\end{align*}
\]

Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?

The Forward/Backward/Forward (FBF) Algorithm

- Facts often have many derivations, so many facts get deleted in the first step
- The Forward/Backward/Forward algorithm looks for alternatives immediately

\[
\begin{array}{c|c}
A(a) & \times \\
B(a) & \quad \text{Delete } A(a) \text{ using FBF:} \\
C_0(a) & \\
C_1(a) & \\
\ldots & \\
C_n(a) & \\
\end{array}
\]

1. Is \( A(a) \) derivable in any other way?
2. No \( \Rightarrow \) delete

The Forward/Backward/Forward Algorithm

- Facts often have many derivations, so many facts get deleted in the first step.
- The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?
2. No $\Rightarrow$ delete
3. As in DRed, identify $C_0(a)$ as derivable from $A(a)$

---

The Forward/Backward/Forward (FBF) Algorithm

Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?
2. No $\Rightarrow$ delete
3. As in DRed, identify $C_0(a)$ as derivable from $A(a)$
4. Apply the rules to $C_0(a)$ ‘backwards’ $\Rightarrow$ by $C_0(x) \leftarrow B(x)$, we get $B(a)$

The Forward/Backward/Forward (FBF) Algorithm

Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?
2. No ⇒ delete
3. As in DRed, identify $C_0(a)$ as derivable from $A(a)$
4. Apply the rules to $C_0(a)$ ‘backwards’ ⇒ by $C_0(x) ← B(x)$, we get $B(a)$
5. $B(a)$ is explicit so it is derivable

The Forward/Backward/Forward (FBF) Algorithm

Facts often have many derivations, so many facts get deleted in the first step.

The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?
2. No ⇒ delete
3. As in DRed, identify $C_0(a)$ as derivable from $A(a)$
4. Apply the rules to $C_0(a)$ ‘backwards’ ⇒ by $C_0(x) \leftarrow B(x)$, we get $B(a)$
5. $B(a)$ is explicit so it is derivable
6. So $C_0(a)$ is derivable too

The Forward/Backward/Forward (FBF) Algorithm

- Facts often have many derivations, so many facts get deleted in the first step.
- The Forward/Backward/Forward algorithm looks for alternatives immediately.

Delete $A(a)$ using FBF:

1. Is $A(a)$ derivable in any other way?
2. No $\Rightarrow$ delete
3. As in DRed, identify $C_0(a)$ as derivable from $A(a)$
4. Apply the rules to $C_0(a)$ ‘backwards’ $\Rightarrow$ by $C_0(x) \leftarrow B(x)$, we get $B(a)$
5. $B(a)$ is explicit so it is derivable
6. So $C_0(a)$ is derivable too
7. Stop propagation and terminate

# Table of Contents

1. **Incremental Reasoning in Datalog**
2. **Counting-Based Algorithms**
3. **The Delete/Rederive (DRed) Algorithm**
4. **The Forward/Backward/Forward (FBF) Algorithm**
5. **Comparing the Three Algorithms**
6. **Extending Datalog: Stratified Negation and Equality**
7. **Other Approaches: FOIESs & Truth Maintenance Systems**
8. **Conclusion**
COMPARISON: RECURSIVE COUNTING

EXAMPLE PROGRAM + INITIAL DATA

\[ A(y) \leftarrow A(x) \land R(x, y), \quad A(a_0), R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n) \]

- Initial state
  
  \[
  \begin{array}{c|c}
    A(a_0) & 1 \\
    A(a_1) & 1 \\
    \vdots & \vdots \\
    A(a_i) & 1 \\
    A(a_{i+1}) & 1 \\
    \vdots & \vdots \\
    A(a_n) & 1 \\
  \end{array}
  \]

- After adding \( A(a_i) \)
  
  \[
  \begin{array}{c|c}
    A(a_0) & 1 \\
    A(a_1) & 1 \\
    \vdots & \vdots \\
    A(a_i) & 1 \\
    A(a_{i+1}) & 1 \\
    \vdots & \vdots \\
    A(a_n) & 1 \\
  \end{array}
  \]

- Even if nothing changes, we can redo initial work twice (if \( i = 2 \))
  - Not worst-case optimal

- The amount of repeated work gets smaller as \( i \) approaches \( n \)
  - \( \Rightarrow \) Intuition: long multiple inference chains can be problematical

- Can be made weakly worst-case optimal
Comparing the Three Algorithms

## Comparison: DRed and FBF

### Example Program + Initial Data

\[
A(y) \leftarrow A(x) \land R(x, y), \quad A(a_0), A(a_i), R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)
\]

- Update task: delete \(A(a_i)\)

- DRed deletes and then reproves \(A(a_j)\) with \(j \geq i\)
- More efficient if \(j\) is close to \(n\)
- Suitable when deletion is at the ‘end’ of inference chains
- Can be made weakly worst-case optimal

- FBF reproves \(A(a_j)\) with \(1 \leq j \leq i\)
- More efficient if \(j\) is close to 0
- Suitable when deletion is at the ‘front’ of inference chains
- Works well if facts can be reproved easily
- Not weakly worst-case optimal: ‘old’ rule instances examined once more in backward chaining
# Table of Contents

1. **Incremental Reasoning in Datalog**

2. **Counting-Based Algorithms**

3. **The Delete/Rederive (DRed) Algorithm**

4. **The Forward/Backward/Forward (FBF) Algorithm**

5. **Comparing the Three Algorithms**

6. **Extending Datalog: Stratified Negation and Equality**

7. **Other Approaches: FOIESs & Truth Maintenance Systems**

8. **Conclusion**
### Example Program + Initial Data

\[ B(x) \leftarrow U(x) \land \neg A(x), \quad C(x) \leftarrow U(x) \land \neg B(x), \quad D(x) \leftarrow U(x) \land \neg C(x), \quad U(a) \]

- **Initial state**
  - \[ B(a) \]
  - \[ D(a) \]

- **After adding** \[ A(a) \]
  - \[ A(a) \]
  - \[ C(a) \]

- Deletion and insertion are **not independent**:
  - Insertion into negated atoms is deletion and vice versa

- \[ \Rightarrow \] Must apply deletion and insertion per **stratum**

- Considerable source of complexity for DRed and FBF:
  - Cannot update materialisation in the end
  - For each stratum, both ‘old’ and ‘new’ materialisation must be kept
  - Ensuring **nonrepetition** is not straightforward
### Example Program + Initial Data

\[ y_1 \approx y_2 \leftarrow R(x, y_1) \land R(x, y_2), \quad C(x) \leftarrow A(x) \land B(x), \quad R(c, a), R(c, b), A(a), B(b) \]

- **Materialisation**
  - \( R(c, a) \)
  - \( R(c, b) \)
  - \( a \approx b \)
  - \( A(a) \)
  - \( A(b) \)
  - \( B(a) \)
  - \( B(b) \)
  - \( C(a) \)
  - \( C(b) \)

- **Eliminate redundancy by keeping representatives only:**
  - Representative of \( a \) is \( a \)
  - Representative of \( b \) is \( a \)
  - Representative of \( c \) is \( c \)
  - \( R(c, a) \)
  - \( a \approx a \)
  - \( A(a) \)
  - \( B(a) \)
  - \( C(a) \)
**Example Program + Initial Data**

\[ y_1 \approx y_2 \leftarrow R(x, y_1) \land R(x, y_2), \quad C(x) \leftarrow A(x) \land B(x), \quad R(c, a), R(c, b), A(a), B(b) \]

- Initial representation
  
  \[
  \begin{array}{l}
  R(c, a) \\
  a \approx a \\
  A(a) \\
  B(a) \\
  C(a)
  \end{array}
  \]

- Representation after deleting \( R(c, b) \)
  
  \[
  \begin{array}{l}
  R(c, a) \\
  A(a) \\
  B(b)
  \end{array}
  \]

- \( \Rightarrow \) Deletion may require addition — \( B(b) \) in this case

- In fact, the representation after deletion can have more facts!
Datalog with Equality (III)

DRed algorithm not very efficient due to equality replacement rules:
- \( A(x) \leftarrow A(y) \land x \approx y, \quad B(x) \leftarrow B(y) \land x \approx y, \ldots \)
- Deleting \( a \approx b \) results in deleting all facts containing \( a \) or \( b \)
- Equality rules are highly recursive — the problematical case for DRed
- \( \Rightarrow \) Often overdeletes most of the materialisation

FBF is much more efficient:
- Eager reproving particularly effective with equality
- Can efficiently handle small updates, particularly if equality does not change
# Table of Contents

1. **Incremental Reasoning in Datalog**

2. **Counting-Based Algorithms**

3. **The Delete/Rederive (DRed) Algorithm**

4. **The Forward/Backward/Forward (FBF) Algorithm**

5. **Comparing the Three Algorithms**

6. **Extending Datalog: Stratified Negation and Equality**

7. **Other Approaches: FOIESs & Truth Maintenance Systems**

8. **Conclusion**
First-Order Incremental Evaluation Systems (FOIESs)

- For single-tuple changes, update the materialisation using a finite number of first-order queries
  - I.e., compute updates using SQL!

Example Program

\[ R(x, z) \leftarrow R(x, y) \land R(y, z) \]
**First-Order Incremental Evaluation Systems (FOIESs)**

- For single-tuple changes, update the materialisation using a finite number of first-order queries
  - I.e., compute updates using SQL!

**Example Program**

\[
R(x, z) \leftarrow R(x, y) \land R(y, z)
\]

Add \( R_{new}(c, d) \)

![Diagram showing the update process with nodes a, b, c, d, e, f connected by relations R and R_{new}]

Boris Motik
A Survey of Incremental Reasoning
First-Order Incremental Evaluation Systems (FOIESs)

- For single-tuple changes, update the materialisation using a finite number of first-order queries
  - I.e., compute updates using SQL!

Example Program

\[ R(x, z) \leftarrow R(x, y) \land R(y, z) \]

- Add \( R_{\text{new}}(c, d) \)
- Evaluate \( R(x, y) \land R_{\text{new}}(y, z) \) and add \( R(x, z) \)
First-Order Incremental Evaluation Systems (FOIESs)

- For single-tuple changes, update the materialisation using a finite number of first-order queries
- I.e., compute updates using SQL!

**Example Program**

\[ R(x, z) \leftarrow R(x, y) \land R(y, z) \]

- Add \( R_{new}(c, d) \)
- Evaluate \( R(x, y) \land R_{new}(y, z) \) and add \( R(x, z) \)
- Evaluate \( R_{new}(x, y) \land R(y, z) \) and add \( R(x, z) \)
**First-Order Incremental Evaluation Systems (FOIESs)**

- For single-tuple changes, update the materialisation using a finite number of first-order queries
  - I.e., compute updates using SQL!

---

**Example Program**

\[ R(x, z) \leftarrow R(x, y) \land R(y, z) \]

---

- Add \( R_{\text{new}}(c, d) \)
- Evaluate \( R(x, y) \land R_{\text{new}}(y, z) \) and add \( R(x, z) \)
- Evaluate \( R_{\text{new}}(x, y) \land R(y, z) \) and add \( R(x, z) \)
- Evaluate \( R(x, y) \land R_{\text{new}}(y, z) \land R(z, w) \) and add \( R(x, w) \)
CAPABILITIES OF FOIESs

- Possible only in limited situations:
  - Adding (but not deleting) tuples on chain programs
  - Deleting tuples for transitivity over *acyclic* data
  - No FOIES exists for transitivity and *cyclic* graphs

- Capabilities increase if we can maintain additional *auxiliary* relations
  - Still cannot handle transitivity over cyclic graphs for deletion

- ⇒ Neat idea, but not really suitable for general use
Truth Maintenance Systems (TMSs)

- **Justification**: a set of facts and a rule that prove some fact
  - Rule $A(x) \leftarrow B(x) \land C(x)$ and facts $B(a)$ and $C(a)$ justify $A(a)$
  - Can be seen as a rule instance $A(a) \leftarrow B(a) \land C(a)$

- Main idea: store with each fact a set of justifications
  - Add a justification each time a fact is derived
  - Delete a justification each time a rule instance goes away
  - Delete a fact when its set of justifications drops to zero

- Main problem: storing justifications incurs significant overhead
  - Particularly on highly recursive programs
  - $\Rightarrow$ Unlikely to be efficient enough for large knowledge bases
# Table of Contents

1. **Incremental Reasoning in Datalog**

2. **Counting-Based Algorithms**

3. **The Delete/Rederive (DRed) Algorithm**

4. **The Forward/Backward/Forward (FBF) Algorithm**

5. **Comparing the Three Algorithms**

6. **Extending Datalog: Stratified Negation and Equality**

7. **Other Approaches: FOIESs & TruthMaintenance Systems**

8. **Conclusion**
Three algorithms most suitable for practice:
  - Counting (with a nontrivial extension to recursion)
  - DRed
  - FBF

No algorithm is worst-case optimal
  - Can in fact be twice as inefficient as rematerialisation

Counting and DRed are optimal in a weak sense
  - They consider at most all inferences from the ‘old’ and the ‘new’ derivation

FBF is not weakly optimal due to backward chaining

Counting has a fixed overhead that shows up on small updates

FBF seems to be particularly suited to small updates