Stop the Chase

Michael Meier, Michael Schmidt, and Georg Lausen

Freiburg University Database Group

Alberto Mendelzon Workshop on Foundations of Data Management
May 12, 2009
The Chase Algorithm

- Central tool in many database areas, such as Semantic Query Optimization, Query Rewriting using Views, Data Integration ...
- Idea: given a database instance and a set of constraints, fix the constraint violations in the database instance
Motivation

The Chase Algorithm

- Central tool in many database areas, such as Semantic Query Optimization, Query Rewriting using Views, Data Integration ...
- Idea: given a database instance and a set of constraints, fix the constraint violations in the database instance

Chase Termination

- The Chase Algorithm does not necessarily terminate
- Even worse: Chase termination is an undecidable problem in general, even for a fixed instance
- *Sufficient* conditions over the constraints exist, which guarantee Chase termination on every instance
Contributions and Outline

Novel Data-independent Chase Termination Conditions

- Apply to every database instance
- Generalize previous conditions, such as *Weak Acyclicity* and *Stratification*
- Allows us to guarantee Chase termination in more cases

Study of Data-dependent Chase Termination

- Static approach
- Dynamic approach
A Non-terminating Chase Sequence

Example

Consider a single predicate $E(src, dest)$ (storing graph edges), the database instance $I$, and constraint $\alpha$:

$$I := \{E(a, b)\} \quad \quad \alpha := \forall x, y(E(x, y) \rightarrow \exists z(E(y, z)))$$

The Chase Algorithm tries to fix constraint violations in $I$:

$$\begin{align*}
\{E(a, b)\} & \xrightarrow{\alpha} \{E(a, b), E(b, n_1)\}, \text{ where } n_1 \text{ is a fresh null value} \\
\{E(a, b), E(b, n_1)\} & \xrightarrow{\alpha} \{E(a, b), E(b, n_1), E(n_1, n_2)\}, \text{ where } n_2 \text{ is a fresh null value} \\
\{E(a, b), E(b, n_1), E(n_1, n_2)\} & \xrightarrow{\alpha} \{E(a, b), E(b, n_1), E(n_1, n_2), E(n_2, n_3)\}, \text{ where } n_3 \text{ is a fresh null value} \\
\vdots
\end{align*}$$

Michael Meier, Michael Schmidt, and Georg Lausen

Stop the Chase
A Non-terminating Chase Sequence

Example

Consider a single predicate \( E(src, dest) \) (storing graph edges), the database instance \( \mathcal{I} \), and constraint \( \alpha \):

\[
\mathcal{I} := \{ E(a, b) \} \quad \quad \alpha := \forall x, y (E(x, y) \rightarrow \exists z (E(y, z)))
\]

The Chase Algorithm tries to fix constraint violations in \( \mathcal{I} \):

\[
\begin{align*}
\{ E(a, b) \} & \xrightarrow{\alpha} \{ E(a, b), E(b, n_1) \}, \text{ where } n_1 \text{ is a fresh null value} \\
\{ E(a, b), E(b, n_1) \} & \xrightarrow{\alpha} \{ E(a, b), E(b, n_1), E(n_1, n_2) \}, \text{ where } n_2 \text{ is a fresh null value} \\
\{ E(a, b), E(b, n_1), E(n_1, n_2) \} & \xrightarrow{\alpha} \{ E(a, b), E(b, n_1), E(n_1, n_2), E(n_2, n_3) \}, \text{ where } n_3 \text{ is a fresh null value} \\
& \ldots
\end{align*}
\]

Source of Non-termination

Cascading of fresh null values (here: created in position \( E^2 \)).
Central Ideas

Estimating the Flow of Null Values

- Estimate positions where null values can be created in or copied to during the Chase run
- Supervise the flow of null values
- More fine-grained decomposition of the constraint set than in previous conditions
Survey of Results

Inductive Restriction

Safe Restriction

Stratification

Safety

Weak Acyclicity
The Limitations of Weak Acyclicity

**Weak Acyclicity**
- Construct the *dependency graph*, which tracks the flow of values.
- Use “special edges” $\rightarrow^*$ where fresh null values are created.

$$\forall x, y (E(x, y) \rightarrow \exists z E(y, z))$$

Dependency graph:

```
E^1 \rightarrow E^2
```

$$\forall x, y (S(y), E(x, y) \rightarrow \exists z E(y, z))$$

Dependency graph:

```
E^1 \rightarrow E^2
S^1 \rightarrow^* \rightarrow^* \rightarrow^* E^2
```

---

Michael Meier, Michael Schmidt, and Georg Lausen
Stop the Chase
The Limitations of Weak Acyclicity

**Weak Acyclicity**
- Construct the *dependency graph*, which tracks the flow of values
- Use “special edges” $\rightarrow^*$ where fresh null values are created

$$\forall x, y (E(x, y) \rightarrow \exists z E(y, z))$$

Dependency graph:

```
E^1 \rightarrow E^2
```

$$\forall x, y (S(y), E(x, y) \rightarrow \exists z E(y, z))$$

Dependency graph:

```
E^1 \rightarrow E^2
\text{S}^1 \rightarrow^*\rightarrow^*
```

"Unnatural" Property of Weak Acyclicity
More literals in the body imply larger dependency graph.
Estimating Positions with Null Values

Estimating Positions That May Contain Null Values

- Borrow the notion of *affected positions* from [1]
- *Affected positions* are an overestimation of the positions where null values might be created in or copied to during Chase application

Reference

Estimating Positions with Null Values

Estimating Positions That May Contain Null Values

- Borrow the notion of *affected positions* from [1]
- *Affected positions* are an overestimation of the positions where null values might be created in or copied to during Chase application

Reference


Example

\[ \forall x, y (E(x, y) \rightarrow \exists z E(y, z)) \quad \rightarrow \quad \text{Affected positions: } \{E^1, E^2\} \]

\[ \forall x, y (S(y), E(x, y) \rightarrow \exists z E(y, z)) \quad \rightarrow \quad \text{Affected positions: } \{E^2\} \]
The Propagation Graph

- Strict generalization of the Dependency Graph
- Takes affected positions into account
The Propagation Graph

Properties of the Propagation Graph

- Strict generalization of the Dependency Graph
- Takes affected positions into account

\[ \forall x, y (E(x, y) \rightarrow \exists z E(y, z)) \]

Affected positions: \( \{E^1, E^2\} \)

Propagation graph:

\[ E^1 \rightarrow E^2 \]

\[ \forall x, y (S(y), E(x, y) \rightarrow \exists z E(y, z)) \]

Affected positions: \( \{E^2\} \)

Propagation graph:

\[ E^2 \]
The Safety Condition

Definition

A constraint set is called safe iff its propagation graph contains no cycle going through a special edge.
**The Safety Condition**

**Definition**
A constraint set is called *safe* iff its propagation graph contains no cycle going through a special edge.

**Properties of Safety**
- *Safety* guarantees Chase termination in polynomial-time data complexity.
- It can be checked in polynomial time if a constraint set is *safe*.
- *Safety* is strictly more general than *Weak Acyclicity*.

Michael Meier, Michael Schmidt, and Georg Lausen
Limitations of Stratification

- **Stratification** [2]: it suffices to assert *Weak Acyclicity* locally, for subsets of constraints that might cyclically fire each other.

**Reference**

Limitations of Stratification

- *Stratification* [2]: it suffices to assert *Weak Acyclicity* locally, for subsets of constraints that might cyclically fire each other.

**Reference**


**Example**

The following constraint set $\Sigma := \{\alpha_1, \alpha_2\}$ is neither *Weakly Acyclic*, nor *Safe*, nor *Stratified*.

- $\alpha_1$: Special nodes have a cycle of length 2 through outgoing edges
  \[ \forall x, y(S(x), E(x, y) \rightarrow E(y, x)) \]

- $\alpha_2$: Special nodes have a cycle of length 3 through outgoing edges
  \[ \forall x, y(S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x)) \]
### Limitations of Stratification

<table>
<thead>
<tr>
<th>Example (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_1$: Special nodes have a cycle of length 2 through outgoing edges</td>
</tr>
<tr>
<td>$\forall x, y (S(x), E(x, y) \rightarrow E(y, x))$</td>
</tr>
<tr>
<td>$\alpha_2$: Special nodes have a cycle of length 3 through outgoing edges</td>
</tr>
<tr>
<td>$\forall x, y (S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x))$</td>
</tr>
</tbody>
</table>
Limitations of Stratification

Example (continued)

\[ \alpha_1: \text{Special nodes have a cycle of length 2 through outgoing edges} \]
\[ \forall x, y (S(x), E(x, y) \rightarrow E(y, x)) \]

\[ \alpha_2: \text{Special nodes have a cycle of length 3 through outgoing edges} \]
\[ \forall x, y (S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x)) \]

Firing of Constraints

- Firing \( \alpha_1 \) cannot cause \( \alpha_1 \) to fire
- Firing \( \alpha_2 \) cannot cause \( \alpha_2 \) to fire
Limitations of Stratification

Example (continued)

\[ \alpha_1 : \text{Special nodes have a cycle of length 2 through outgoing edges} \]
\[ \forall x, y(S(x), E(x, y) \rightarrow E(y, x)) \]

\[ \alpha_2 : \text{Special nodes have a cycle of length 3 through outgoing edges} \]
\[ \forall x, y(S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x)) \]

Firing of Constraints

- Firing \( \alpha_1 \) can cause \( \alpha_2 \) to fire:
  \[ \{S(a), S(b), E(a, b), E(b, c), E(c, a)\} \]
  \[ \xrightarrow{\alpha_1} \{S(a), S(b), E(a, b), E(b, c), E(c, a), E(b, a)\} \]
  \[ \xrightarrow{\alpha_2} \{S(a), S(b), E(a, b), E(b, c), E(c, a), E(b, a), E(a, n_1), E(n_1, b)\} \]

- Firing \( \alpha_2 \) can cause \( \alpha_1 \) to fire (similar)
Limitations of Stratification

Example (continued)

\( \alpha_1 : \) Special nodes have a cycle of length 2 through outgoing edges
\[ \forall x, y(S(x), E(x, y) \rightarrow E(y, x)) \]

\( \alpha_2 : \) Special nodes have a cycle of length 3 through outgoing edges
\[ \forall x, y(S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x)) \]

Firing of Constraints

- Firing \( \alpha_1 \) can cause \( \alpha_2 \) to fire:
  \[ \{ S(a), S(b), E(a, b), E(b, c), E(c, a) \} \]
  \[ \xrightarrow{\alpha_1} \{ S(a), S(b), E(a, b), E(b, c), E(c, a), E(b, a) \} \]
  \[ \xrightarrow{\alpha_2} \{ S(a), S(b), E(a, b), E(b, c), E(c, a), E(b, a), E(a, n_1), E(n_1, b) \} \]
- Firing \( \alpha_2 \) can cause \( \alpha_1 \) to fire (similar)

\[ \rightarrow \text{Constraints are cyclically connected, so Stratification does not apply.} \]
Decomposition of the Constraint Set

Observation

Not only cyclic firing, but also a cyclic passing of null values is necessary to obtain non-terminating Chase sequences.
Observation

Not only cyclic firing, but also a cyclic passing of null values is necessary to obtain non-terminating Chase sequences.

Example (continued)

\[\alpha_1: \text{Special nodes have a cycle of length 2 through outgoing edges} \]
\[\forall x, y (S(x), E(x, y) \rightarrow E(y, x))\]

\[\alpha_2: \text{Special nodes have a cycle of length 3 through outgoing edges} \]
\[\forall x, y (S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x))\]

- Firing \(\alpha_1\) cannot cause \(\alpha_2\) to fire s.t. \(\alpha_2\) copies some null values from its head to its body, since \(S^1\) does not contain null values.
Safe Restriction

Refinement of Firing

- Ignore firing relation of \((\alpha_1, \alpha_2)\) whenever the firing of \(\alpha_2\) (after firing \(\alpha_1\)) cannot copy null values from its body to its head
- Allows for more fine-grained decomposition of the constraint set
- Check Safety condition for the decomposed sets

→ new Chase termination condition called *Safe Restriction*
Safe Restriction

Refinement of Firing

- Ignore firing relation of \((\alpha_1, \alpha_2)\) whenever the firing of \(\alpha_2\) (after firing \(\alpha_1\)) cannot copy null values from its body to its head.
- Allows for more fine-grained decomposition of the constraint set.
- Check Safety condition for the decomposed sets.

→ new Chase termination condition called **Safe Restriction**

**Safe Restriction**

- *Safe Restriction* guarantees Chase termination in polynomial time data complexity.
- It can be checked by a \textsc{coNP}-algorithm if a constraint set is safely restricted.
- *Safe Restriction* strictly generalizes *Stratification*.
Inductive Restriction

\( \alpha_1 \): Every special node with an edge has a cycle of length 2
\[ \forall x, y (S(x), E(x, y) \rightarrow E(y, x)) \]

\( \alpha_2 \): Every special with and edge has a cycle of length 3
\[ \forall x, y (S(x), E(x, y) \rightarrow \exists z E(y, z), E(z, x)) \]

\( \alpha_3 \): There is at least one special node with outgoing edge
\[ \exists x, y S(x), E(x, y) \]

**Inductive Restriction**

Now a cyclic passing of null values between \( \alpha_2 \) and \( \alpha_1 \) becomes possible, because there might occur null values in \( S^1 \) now.

Idea: inductive decomposition of the constraint set gives us a novel termination called **Inductive Restriction**, which further generalizes **Safe Restriction**.
Chase termination w.r.t. a fixed instance

- May overcome situations where no termination guarantees for the general case can be made
- Given a constraint set $\Sigma$ and a database instance $I$, we propose two complementary approaches
  - Static approach
  - Dynamic approach
Static Approach

Idea

- Try to exclude constraints from the constraint set that will never fire when chasing the instance under consideration.
- Check if data-independent termination condition (i.e., *Inductive Restriction*) holds for this subset of relevant constraints.
### Static Approach

#### Idea
- Try to exclude constraints from the constraint set that will never fire when chasing the instance under consideration.
- Check if data-independent termination condition (i.e., *Inductive Restriction*) holds for this subset of relevant constraints.

#### Challenge
- Fundamental result: it is an undecidable problem if a constraint fires when chasing a fixed instance.
- Techniques to overestimate the constraint set that may be used during the Chase on the instance.
Dynamic Approach

**Monitoring Chase Execution**

- Maintain a data structure (called *monitor graph*) that tracks the repeated introduction of fresh null values.
- Fix a repetition threshold $k$ and abort the Chase run if $k$ is exceeded: in such a case, no termination guarantees can be made.
Dynamic Approach

Monitoring Chase Execution

- Maintain a data structure (called *monitor graph*) that tracks the repeated introduction of fresh null values.
- Fix a repetition threshold \( k \) and abort the Chase run if \( k \) is exceeded: in such a case, no termination guarantees can be made.

Properties of the Monitoring Approach

- **Guarantee**: infinite Chase sequences will always be detected, independent from the size of the repetition threshold.
- **Natural condition**: the monitor graph accounts for situations that may well cause non-termination.
- **Pay-as-you-go approach**: the repetition threshold can be chosen following a pay-as-you-go approach: The higher the threshold, the more terminating Chase sequences will be recognized.
**Data-independent Chase Termination**

- Estimation of the positions that may contain null values and tracking the flow of null values allows us to improve existing sufficient termination conditions
  - *Safety* (checkable in polynomial time)
  - *Safe Restriction* and *Inductive Restriction* (checkable by a coNP-algorithm)

**Data-dependent Chase Termination**

- First results on Chase termination for fixed instances
  - Static approach: exclude irrelevant constraints
  - Dynamic approach: monitor Chase at runtime
Thank You for Your Attention!

Any questions?