Evaluation of Datalog Programs

July 5, 2018
What is Evaluation?

Datalog program consists of base facts and rules.

Evaluation: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining.

Forward Chaining: Iteratively apply rules on the set of currently known facts.
If A is the case, and A implies B, then B is the case.
If B implies C...

Backward Chaining: Starting from a query, search for rule heads or known facts which produce it.
Is B true?
If A implies B, then for B to be true it suffices to show that A holds.
If C implies A...
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining
Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
What is Evaluation?

Datalog program consists of base facts and rules.

Evaluation: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  
  *If A is the case, and A implies B, then B is the case.*
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  
  *If A is the case, and A implies B, then B is the case.*
  
  *If B implies C...*
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  - *If A is the case, and A implies B, then B is the case.*
  - *If B implies C*...

- **Backward Chaining**: Starting from a query, search for rule heads or known facts which produce it.
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  
  *If A is the case, and A implies B, then B is the case.*
  
  *If B implies C...*

- **Backward Chaining**: Starting from a query, search for rule heads or known facts which produce it.
  
  *Is B true?*
Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  
  *If A is the case, and A implies B, then B is the case."
  
  *If B implies C..."

- **Backward Chaining**: Starting from a query, search for rule heads or known facts which produce it.
  
  *Is B true?"
  
  *If A implies B, then for B to be true it suffices to show that A holds."
What is Evaluation?

Datalog program consists of base facts and rules.

**Evaluation**: Finding out which facts are derivable from base facts given the rules.

Two general ways of evaluation: Forward Chaining & Backward Chaining

- **Forward Chaining**: Iteratively apply rules on the set of currently known facts.
  
  *If A is the case, and A implies B, then B is the case.*
  
  *If B implies C...*

- **Backward Chaining**: Starting from a query, search for rule heads or known facts which produce it.
  
  *Is B true?*
  
  *If A implies B, then for B to be true it suffices to show that A holds.*
  
  *If C implies A...*
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**
- Is exhaustive in producing all derivable facts
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**

- is exhaustive in producing all derivable facts
- terminates:

Let $|\text{objects}(P)| = N$, $|\text{predicate symbols}(P)| = p < \infty$, then $K := \max_{Q'} \text{arity}(Q')$. Let $Q_{\text{max}}$ be the predicate with the highest arity. $K$ objects can be inserted into $Q_{\text{max}}$, i.e., $Q_{\text{max}}(a_1, ..., a_K)$. For any predicate symbol $Q$, at most $N^K$ objects can be inserted. Therefore, $|\text{facts}| \leq p \cdot N^K$. Not goal directed/potentially computes irrelevant facts.
Forward Chaining:

+ is exhaustive in producing all derivable facts
+ terminates:

Any Datalog program $P$ consists of finitely many base facts and rules
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**
- is exhaustive in producing all derivable facts
- terminates:
  - Any Datalog program $P$ consists of finitely many base facts and rules
  - number of objects & predicate symbols occurring in $P$ is finite
Practicality of Forward resp. Backward Chaining

Forward Chaining:
+ is exhaustive in producing all derivable facts
+ terminates:
  Any Datalog program $P$ consists of finitely many base facts and rules
  $\implies$ number of objects & predicate symbols occurring in $P$ is finite
  Let $|\text{objects}(P)| = N$ & $|\text{predicate symbols}(P)| = p$
Practicality of Forward resp. Backward Chaining

Forward Chaining:
- is exhaustive in producing all derivable facts
- terminates:
  Any Datalog program $P$ consists of finitely many base facts and rules
  $\implies$ number of objects & predicate symbols occurring in $P$ is finite
  Let $|\text{objects}(P)| = N$ & $|\text{predicate symbols}(P)| = p$

  \[ p < \infty \implies K := \max_{\text{pred. symbol}} \text{arity}(Q') \in \mathbb{N} \]
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**

+ is exhaustive in producing all derivable facts

+ terminates:

Any Datalog program $P$ consists of finitely many base facts and rules

$\implies$ number of objects & predicate symbols occurring in $P$ is finite

Let $\mid \text{objects}(P) \mid = N$ & $\mid \text{predicate symbols}(P) \mid = p$

$p \lt \infty \implies K := \max_{Q' \text{ pred. symbol}} \text{arity}(Q') \in \mathbb{N}$

Let $Q_{\text{max}}$ be predicate with highest arity $K$:
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**
- is exhaustive in producing all derivable facts
- terminates:
  
  Any Datalog program $P$ consists of finitely many base facts and rules
  
  $\implies$ number of objects & predicate symbols occurring in $P$ is finite
  
  Let $|\text{objects}(P)| = N \& |\text{predicate symbols}(P)| = p$

  $$p < \infty \implies K := \max_{Q' \text{ pred. symbol}} \text{arity}(Q') \in \mathbb{N}$$

  Let $Q_{\text{max}}$ be predicate with highest arity $K$:
  
  $N^K$ ways to insert objects into this predicate:
  
  $Q_{\text{max}}(a_1, \ldots, a_1), Q_{\text{max}}(a_1, \ldots, a_2), \ldots, Q_{\text{max}}(a_N, \ldots, a_N)$
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**

- is exhaustive in producing all derivable facts

- terminates:

  Any Datalog program $P$ consists of finitely many base facts and rules
  $\implies$ number of objects & predicate symbols occurring in $P$ is finite

  Let $|\text{objects}(P)| = N$ & $|\text{predicate symbols}(P)| = p$

  $p < \infty \implies K := \max_{Q' \text{ pred. symbol}} \text{arity}(Q') \in \mathbb{N}$

  Let $Q_{\text{max}}$ be predicate with highest arity $K$:
  $N^K$ ways to insert objects into this predicate:
  $Q_{\text{max}}(a_1, ..., a_1), Q_{\text{max}}(a_1, ..., a_2), ..., Q_{\text{max}}(a_N, ..., a_N)$
  $\implies$ for any predicate symbol $Q$, at most $N^K$ ways to insert objects

  $|\text{facts}| \leq p \cdot N^K$ not goal directed/potentially computes irrelevant facts
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**
+ is exhaustive in producing all derivable facts
+ terminates:
  Any Datalog program $P$ consists of finitely many base facts and rules
  $\implies$ number of objects & predicate symbols occurring in $P$ is finite
  Let $|\text{objects}(P)| = N$ & $|\text{predicate symbols}(P)| = p$

$$p < \infty \implies K := \max_{Q': \text{pred. symbol}} \text{arity}(Q') \in \mathbb{N}$$

Let $Q_{\max}$ be predicate with highest arity $K$:
$N^K$ ways to insert objects into this predicate:
$Q_{\max}(a_1, \ldots, a_1), Q_{\max}(a_1, \ldots, a_2), \ldots, Q_{\max}(a_N, \ldots, a_N)$
$\implies$ for any predicate symbol $Q$, at most $N^K$ ways to insert objects
$\implies |\text{facts}| \leq p \cdot N^K$
Practicality of Forward resp. Backward Chaining

**Forward Chaining:**

+ is exhaustive in producing all derivable facts
+ terminates:
  
  Any Datalog program $P$ consists of finitely many base facts and rules
  \[ \implies \text{number of objects \\& predicate symbols occurring in } P \text{ is finite} \]

Let $|\text{objects}(P)| = N$ \\
& $|\text{predicate symbols}(P)| = p$

\[ p < \infty \implies K := \max_{Q' \text{ pred. symbol}} \text{arity}(Q') \in \mathbb{N} \]

Let $Q_{\text{max}}$ be predicate with highest arity $K$:

$N^K$ ways to insert objects into this predicate:

$Q_{\text{max}}(a_1, \ldots, a_1), Q_{\text{max}}(a_1, \ldots, a_2), \ldots, Q_{\text{max}}(a_N, \ldots, a_N)$

\[ \implies \text{for any predicate symbol } Q, \text{ at most } N^K \text{ ways to insert objects} \]

\[ \implies |\text{facts}| \leq p \cdot N^K \]

– not goal directed/potentially computes irrelevant facts
Practicality of Forward resp. Backward Chaining

Backward Chaining:

+ is goal directed/considers only facts which have the potential to answer the initial query

Query: is_object(a)?

To answer this, backward chaining tries to answer is_object(a)?...

→ doesn't terminate

Termination often more important than goal directedness

→ use forward chaining
Practicality of Forward resp. Backward Chaining

**Backward Chaining:**

+ is goal directed/considers only facts which have the potential to answer the initial query

− isn’t guaranteed to terminate/may run into infinite loop:

\[
is_{\text{object}}(X) \leftarrow is_{\text{object}}(X)
\]

Query: \(is_{\text{object}}(a)\)

To answer this, backward chaining tries to answer \(is_{\text{object}}(a)\)...

→ doesn’t terminate

Termination often more important than goal directedness

→ use forward chaining
Practicality of Forward resp. Backward Chaining

**Backward Chaining:**

+ is goal directed/considers only facts which have the potential to answer the initial query

− isn’t guaranteed to terminate/may run into infinite loop:

```prolog
is_object(X) ← is_object(X)
```

Termination often more important than goal directedness
→ use forward chaining
Practicality of Forward resp. Backward Chaining

**Backward Chaining:**

+ is goal directed/considers only facts which have the potential to answer the initial query

− isn’t guaranteed to terminate/may run into infinite loop:

\[
\text{is\_object}(X) \leftarrow \text{is\_object}(X)
\]

Query: \text{is\_object}(a)?
Backward Chaining:

+ is goal directed/considers only facts which have the potential to answer the initial query

- isn’t guaranteed to terminate/may run into infinite loop:

\[
\text{is\_object}(X) \leftarrow \text{is\_object}(X)
\]

Query: \text{is\_object}(a)?
To answer this, backward chaining tries to answer \text{is\_object}(a)?...
Practicality of Forward resp. Backward Chaining

**Backward Chaining:**

+ is goal directed/considers only facts which have the potential to answer the initial query

− isn’t guaranteed to terminate/may run into infinite loop:

\[
\text{is\_object}(X) \leftarrow \text{is\_object}(X)
\]

Query: \text{is\_object}(a)?
To answer this, backward chaining tries to answer \text{is\_object}(a)?...
→ doesn’t terminate
Practicality of Forward resp. Backward Chaining

**Backward Chaining:**

+ is goal directed/considers only facts which have the potential to answer the initial query

  − isn’t guaranteed to terminate/may run into infinite loop:

    ```
    is_object(X) ← is_object(X)
    ```

Query: `is_object(a)?`

To answer this, backward chaining tries to answer `is_object(a)?`...

→ doesn’t terminate

Termination often more important than goal directedness

→ use forward chaining
Immediate Consequence Operator

Immediate Consequence: Known or follows by single application of the rules to the known facts
Immediate Consequence Operator

Immediate Consequence: Known or follows by *single* application of the rules to the *known* facts

The *immediate consequence operator* $T_P$ maps an instance to the set of immediate consequences

$I_0 := \{\text{edge}(a,b)\}$

$I_1 := T_P(I_0) = \{\text{edge}(a,b), \text{path}(a,b)\}$

Iterative forward chaining $\overset{\text{hat}}{=} \text{Repeated application of } T_P$
Immediate Consequence Operator

Immediate Consequence: Known or follows by *single* application of the rules to the *known* facts

The *immediate consequence operator* $TP$ maps an instance to the set of immediate consequences

**Example**

- $\text{edge}(a,b)$
- $\text{path}(X,Y) \leftarrow \text{edge}(X,Y)$

$I_0 := \{\text{edge}(a,b)\}$
Immediate Consequence Operator

Immediate Consequence: Known or follows by single application of the rules to the known facts

The immediate consequence operator $T_P$ maps an instance to the set of immediate consequences

```plaintext
edge(a,b)

path(X,Y) ← edge(X,Y)

I_0 := \{edge(a,b)\}
I_1 := T_P(I_0) = \{edge(a,b), path(a,b)\}
```
Immediate Consequence Operator

Immediate Consequence: Known or follows by *single* application of the rules to the *known* facts

The *immediate consequence operator* $T_P$ maps an instance to the set of immediate consequences

\[
\begin{align*}
\text{edge}(a,b) \\
\text{path}(X,Y) &\leftarrow \text{edge}(X,Y)
\end{align*}
\]

\[
\begin{align*}
l_0 &:= \{\text{edge}(a,b)\} \\
l_1 &:= T_P(l_0) = \{\text{edge}(a,b), \text{path}(a,b)\}
\end{align*}
\]

Iterative forward chaining $\hat{=}$ Repeated application of $T_P$
Immediate Consequence Operator

Immediate Consequence: Known or follows by single application of the rules to the known facts

The immediate consequence operator $T_P$ maps an instance to the set of immediate consequences

\[
\begin{align*}
\text{edge}(a,b) \\
\text{path}(X,Y) &\leftarrow \text{edge}(X,Y) \\
I_0 &:= \{\text{edge}(a,b)\} \\
I_1 &:= T_P(I_0) = \{\text{edge}(a,b), \text{path}(a,b)\}
\end{align*}
\]

Iterative forward chaining $\hat{=}$ Repeated application of $T_P$

We know from the talk on fixpoints that:

- $T_P$ has unique least fixpoint
Immediate Consequence Operator

Immediate Consequence: Known or follows by single application of the rules to the known facts

The immediate consequence operator $T_P$ maps an instance to the set of immediate consequences

\[
\text{edge}(a,b)
\]

\[
\text{path}(X,Y) \leftarrow \text{edge}(X,Y)
\]

\[
l_0 := \{\text{edge}(a,b)\}
\]

\[
l_1 := T_P(l_0) = \{\text{edge}(a,b), \text{path}(a,b)\}
\]

Iterative forward chaining $\hat{=} \text{ Repeated application of } T_P$

We know from the talk on fixpoints that:

- $T_P$ has unique least fixpoint
- this fixpoint corresponds to the unique minimal Herbrand model of $P$
Immediate Consequence Operator

Immediate Consequence: Known or follows by single application of the rules to the known facts

The immediate consequence operator $T_P$ maps an instance to the set of immediate consequences

\[
\begin{align*}
\text{edge}(a,b) \\
\text{path}(X,Y) &\leftarrow \text{edge}(X,Y)
\end{align*}
\]

\[
l_0 := \{\text{edge}(a,b)\} \\
l_1 := T_P(l_0) = \{\text{edge}(a,b), \text{path}(a,b)\}
\]

Iterative forward chaining $\overset{\sim}{=} \text{Repeated application of } T_P$

We know from the talk on fixpoints that:

- $T_P$ has unique least fixpoint
- this fixpoint corresponds to the unique minimal Herbrand model of $P$
- for DL programs reached in finitely many steps (bound on slide 2)
Translation of immediate consequence operator into actual algorithm
Translation of immediate consequence operator into actual algorithm

\[
\begin{align*}
i & := 0 \\
Known_0 & := \text{baseFacts}(P) \\
do\{ \\
\quad & i := i + 1 \\
\quad & \text{Insts} := \text{instantiations(rules}(P), \ Known_{i-1}) \\
\quad & \text{Known}_i := \text{Known}_{i-1} \cup \text{heads} (\text{Insts}) \\
\} \text{ while } \text{Known}_i \neq \text{Known}_{i-1} \\
\text{return } \text{Known}_i
\end{align*}
\]
Naive Evaluation

Translation of immediate consequence operator into actual algorithm

\[
i := 0
\]
\[
Known_0 := \text{baseFacts}(P)
\]
\[
\text{do}\{
\quad i := i + 1
\quad \text{Insts} := \text{instantiations}(\text{rules}(P), \ Known_{i-1})
\quad Known_i := Known_{i-1} \cup \text{heads(Insts)}
\}\quad \text{while} \ Known_i \neq Known_{i-1}
\]
\text{return } Known_i

\textbf{Known}_i \text{ is the set of facts currently known in iteration } i \text{ / the set of immediate consequences of } Known_{i-1}
Naive Evaluation

Translation of immediate consequence operator into actual algorithm

\[
\begin{align*}
i &:= 0 \\
Known_0 &:= \text{baseFacts}(P) \\
do\{ & \\
& \quad i := i + 1 \\
& \quad Insts := \text{instantiations}(\text{rules}(P), Known_{i-1}) \\
& \quad Known_i := Known_{i-1} \cup \text{heads}(Insts) \\
\} \quad \text{while } Known_i \neq Known_{i-1} \\
\text{return } Known_i
\end{align*}
\]

- \(Known_i\) is the set of facts currently known in iteration \(i\) / the set of immediate consequences of \(Known_{i-1}\)
- one run through the loop corresponds to single application of \(TP\)
Naive Evaluation

Translation of immediate consequence operator into actual algorithm

\[ i := 0 \]
\[ \text{Known}_0 := \text{baseFacts}(P) \]
\[ \text{do} \{ \]
\[ \quad i := i + 1 \]
\[ \quad \text{Insts} := \text{instantiations}(\text{rules}(P), \text{Known}_{i-1}) \]
\[ \quad \text{Known}_i := \text{Known}_{i-1} \cup \text{heads}(\text{Insts}) \]
\[ \} \text{while } \text{Known}_i \neq \text{Known}_{i-1} \]
\[ \text{return } \text{Known}_i \]

- \( \text{Known}_i \) is the set of facts currently known in iteration \( i \) / the set of immediate consequences of \( \text{Known}_{i-1} \)
- \( \rightarrow \) one run through the loop corresponds to single application of \( T_P \)
- \( \rightarrow \) algorithm terminates & returns the minimal fixpoint of \( T_P \)
Naive Evaluation Example

Example program CT:

\[
\begin{align*}
\text{edge}(a,b) \\
\text{edge}(b,c) \\
\text{path}(X,Y) &\leftarrow \text{edge}(X,Y) \\
\text{path}(X,Z) &\leftarrow \text{path}(X,Y), \text{edge}(Y,Z) \\
\end{align*}
\]

\[
i := 0 \\
\text{Known}_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}
\]
Example program CT:

\[
\begin{align*}
\text{edge}(a,b) \\
\text{edge}(b,c) \\
\text{path}(X,Y) & \leftarrow \text{edge}(X,Y) \\
\text{path}(X,Z) & \leftarrow \text{path}(X,Y), \text{edge}(Y,Z)
\end{align*}
\]

\[i := 0\]
\[\text{Known}_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}\]
\[i := 1\]
\[\text{Insts} := \{\text{path}(a,b) \leftarrow \text{edge}(a,b), \text{path}(b,c) \leftarrow \text{edge}(b,c)\}\]
\[\text{Known}_1 := \{\text{edge}(a,b), \text{edge}(b,c), \text{path}(a,b), \text{path}(b,c)\}\]
Naive Evaluation Example

Example program CT:

\[
\begin{align*}
\text{edge}(a,b) \\
\text{edge}(b,c) \\
\text{path}(X,Y) & \leftarrow \text{edge}(X,Y) \\
\text{path}(X,Z) & \leftarrow \text{path}(X,Y), \text{edge}(Y,Z)
\end{align*}
\]

\[i := 0\]
\[\text{Known}_0 := \{ \text{edge}(a,b), \text{edge}(b,c) \}\]

\[i := 1\]
\[\text{Insts} := \{ \text{path}(a,b) \leftarrow \text{edge}(a,b), \text{path}(b,c) \leftarrow \text{edge}(b,c) \}\]
\[\text{Known}_1 := \{ \text{edge}(a,b), \text{edge}(b,c), \text{path}(a,b), \text{path}(b,c) \}\]

\[i := 2\]
\[\text{Insts} := \{ \text{path}(a,b) \leftarrow \text{edge}(a,b), \text{path}(b,c) \leftarrow \text{edge}(b,c), \text{path}(a,c) \leftarrow \text{path}(a,b), \text{edge}(b,c) \}\]
\[\text{Known}_2 := \{ \text{edge}(a,b), \text{edge}(b,c), \text{path}(a,b), \text{path}(b,c), \text{path}(a,c) \}\]
Naive Evaluation Example

Example program $CT$:

```plaintext
edge(a,b)
edge(b,c)

path(X,Y) ← edge(X,Y)
path(X,Z) ← path(X,Y), edge(Y,Z)
```

$i := 0$

$Known_0 := \{edge(a,b) , edge(b,c)\}$

$i := 1$

$Insts := \{path(a,b) ← edge(a,b) , path(b,c) ← edge(b,c)\}$

$Known_1 := \{edge(a,b) , edge(b,c) , path(a,b) , path(b,c)\}$

$i := 2$

$Insts := \{path(a,b) ← edge(a,b) , path(b,c) ← edge(b,c) ,$

\hspace{1cm} path(a,c) ← path(a,b) , edge(b,c)\}$

$Known_2 := \{edge(a,b) , edge(b,c) , path(a,b) , path(b,c) , path(a,c)\}$

$i := 3$

$Insts := \{path(a,b) ← edge(a,b) , path(b,c) ← edge(b,c) ,$

\hspace{1cm} path(a,c) ← path(a,b) , edge(b,c)\}$

$Known_3 := \{edge(a,b) , edge(b,c) , path(a,b) , path(b,c) , path(a,c)\}$

return $Known_3$


Naive Evaluation Example

Example program $CT$:

\[
\begin{align*}
\text{edge}(a, b) \\
\text{edge}(b, c)
\end{align*}
\]

\[
\begin{align*}
\text{path}(X, Y) & \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Z) & \leftarrow \text{path}(X, Y), \text{edge}(Y, Z)
\end{align*}
\]

\[
\begin{align*}
i & := 0 \\
\text{Known}_0 & := \{\text{edge}(a, b), \text{edge}(b, c)\}
\end{align*}
\]

\[
\begin{align*}
i & := 1 \\
\text{Insts} & := \{\text{path}(a, b) \leftarrow \text{edge}(a, b), \text{path}(b, c) \leftarrow \text{edge}(b, c)\}
\end{align*}
\]

\[
\begin{align*}
\text{Known}_1 & := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c)\}
\end{align*}
\]

\[
\begin{align*}
i & := 2 \\
\text{Insts} & := \{\text{path}(a, b) \leftarrow \text{edge}(a, b), \text{path}(b, c) \leftarrow \text{edge}(b, c), \text{path}(a, c) \leftarrow \text{path}(a, b), \text{edge}(b, c)\}
\end{align*}
\]

\[
\begin{align*}
\text{Known}_2 & := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}
\end{align*}
\]

\[
\begin{align*}
i & := 3 \\
\text{Insts} & := \{\text{path}(a, b) \leftarrow \text{edge}(a, b), \text{path}(b, c) \leftarrow \text{edge}(b, c), \text{path}(a, c) \leftarrow \text{path}(a, b), \text{edge}(b, c)\}
\end{align*}
\]

\[
\begin{align*}
\text{Known}_3 & := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}
\end{align*}
\]

return \text{Known}_3
Semi Naive Evaluation

- Based on naive evaluation but
Semi Naive Evaluation

- Based on naive evaluation but
- an instantiation now needs to contain at least one fact newly derived in last iteration
Based on naive evaluation but
an instantiation now needs to contain at least one fact newly derived in last iteration

\[
\text{instantiations}_s(\{h(X) ← f(X), g(X)\}, \{f(a), g(a)\}, \emptyset) = \emptyset
\]
whereas
Based on naive evaluation but an instantiation now needs to contain at least one fact newly derived in last iteration

\[ \text{instantiations}_s(\{h(X) \leftarrow f(X), g(X)\}, \{f(a), g(a)\}, \emptyset) = \emptyset \]

whereas

\[ \text{instantiations}_s(\{h(X) \leftarrow f(X), g(X)\}, \{f(a)\}, \{g(a)\}\} = \{h(a) \leftarrow f(a), g(a)\}\]
Semi Naive Evaluation

- Based on naive evaluation but
- an instantiation now needs to contain at least one fact newly derived in last iteration

\[
\text{instantiations}_s(\{h(X) \leftarrow f(X), g(X)\}, \{f(a), g(a)\}, \emptyset) = \emptyset
\]

whereas

\[
\text{instantiations}_s(\{h(X) \leftarrow f(X), g(X)\}, \{f(a)\}, \{g(a)\})
= \{h(a) \leftarrow f(a), g(a)\}
\]

\[
i := 0
\]
\[
\text{Known}_0 := \text{baseFacts}(P)
\]
\[
\text{New}_0 := \text{baseFacts}(P)
\]

while \(\text{New}_i \neq \emptyset\):

\[
i := i + 1
\]
\[
\text{Insts} := \text{instantiations}_s(\text{rules}(P), \text{Known}_{i-1}, \text{New}_{i-1})
\]
\[
\text{New}_i := \text{heads}(\text{Insts})
\]
\[
\text{Known}_i := \text{Known}_{i-1} \cup \text{New}_i
\]

return \(\text{Known}_i\)
Semi Naive Evaluation Example

Example program $CT$:

\[
\begin{align*}
\text{edge}(a,b) \\
\text{edge}(b,c) \\
\text{path}(X,Y) & \leftarrow \text{edge}(X,Y) \\
\text{path}(X,Z) & \leftarrow \text{path}(X,Y), \text{edge}(Y,Z)
\end{align*}
\]

\[i := 0\]
\[\text{KnownFacts}_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}\]
\[\text{NewFacts}_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}\]
Semi Naive Evaluation Example

Example program $CT$:

\[
\begin{align*}
\text{edge}(a,b) \\
\text{edge}(b,c) \\
\text{path}(X,Y) &\leftarrow \text{edge}(X,Y) \\
\text{path}(X,Z) &\leftarrow \text{path}(X,Y),\text{edge}(Y,Z)
\end{align*}
\]

$i := 0$
$KnownFacts_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}$
$NewFacts_0 := \{\text{edge}(a,b), \text{edge}(b,c)\}$

$i := 1$
$Instantiations := \{\text{path}(a,b) \leftarrow \text{edge}(a,b), \text{path}(b,c) \leftarrow \text{edge}(b,c)\}$
$NewFacts_1 := \{\text{path}(a,b), \text{path}(b,c)\}$
$KnownFacts_1 := \{\text{edge}(a,b), \text{edge}(b,c), \text{path}(a,b), \text{path}(b,c)\}$
Example program $CT$:

```prolog
edge(a, b)
edge(b, c)

path(X, Y) ←edge(X, Y)
path(X, Z) ←path(X, Y), edge(Y, Z)
```

$i := 0$

$KnownFacts_0 := \{\text{edge}(a, b), \text{edge}(b, c)\}$

$NewFacts_0 := \{\text{edge}(a, b), \text{edge}(b, c)\}$

$i := 1$

Instantiations := \{path(a, b) ←edge(a, b), path(b, c) ←edge(b, c)\}$

$NewFacts_1 := \{\text{path}(a, b), \text{path}(b, c)\}$

$KnownFacts_1 := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c)\}$

$i := 2$

Instantiations := \{path(a, c) ←path(a, b), \text{edge}(b, c)\}$

$NewFacts_2 := \{\text{path}(a, c)\}$

$KnownFacts_2 := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}$
Example program $CT$:

\[
\begin{align*}
\text{edge}(a, b) \\
\text{edge}(b, c) \\
\text{path}(X, Y) & \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Z) & \leftarrow \text{path}(X, Y), \text{edge}(Y, Z)
\end{align*}
\]

\begin{itemize}
    \item $i := 0$
    \item $\text{KnownFacts}_0 := \{\text{edge}(a, b), \text{edge}(b, c)\}$
    \item $\text{NewFacts}_0 := \{\text{edge}(a, b), \text{edge}(b, c)\}$
    \item $i := 1$
    \item $\text{Instantiations} := \{\text{path}(a, b) \leftarrow \text{edge}(a, b), \text{path}(b, c) \leftarrow \text{edge}(b, c)\}$
    \item $\text{NewFacts}_1 := \{\text{path}(a, b), \text{path}(b, c)\}$
    \item $\text{KnownFacts}_1 := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c)\}$
    \item $i := 2$
    \item $\text{Instantiations} := \{\text{path}(a, c) \leftarrow \text{path}(a, b), \text{edge}(b, c)\}$
    \item $\text{NewFacts}_2 := \{\text{path}(a, c)\}$
    \item $\text{KnownFacts}_2 := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}$
    \item $i := 3$
    \item $\text{Instantiations} := \emptyset$
    \item $\text{NewFacts}_3 := \emptyset$
    \item $\text{KnownFacts}_3 := \{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}$
\end{itemize}

return $\{\text{edge}(a, b), \text{edge}(b, c), \text{path}(a, b), \text{path}(b, c), \text{path}(a, c)\}$
Possible Further Improvements

Semi naive algorithm more efficient than naive algorithm but

→ use 'magic sets': program and query can be transformed into a new program such that for the new program, forward chaining is goal directed with respect to the query

semi naive algorithm throws away information about partial instantiations like

\[ h(a, Y) \leftarrow f(a), g(Y) \]

→ this information could instead be stored to reduce the number of required computations

For advanced settings like a database that changes during evaluation there are specialized methods like 'Incremental Query Evaluation using Conjunctive Queries'
Possible Further Improvements

Semi naive algorithm more efficient than naive algorithm but

- not goal directed $\rightarrow$ use ‘magic sets’:
  program and query can be transformed into a new program such that for the new program, forward chaining is goal directed with respect to the query
Semi naive algorithm more efficient than naive algorithm but

- not goal directed → use 'magic sets': program and query can be transformed into a new program such that for the new program, forward chaining is goal directed with respect to the query

- semi naive algorithm throws away information about partial instantiations like \( h(a, Y) ← f(a), g(Y) \) → this information could instead be stored to reduce the number of required computations
Possible Further Improvements

Semi naive algorithm more efficient than naive algorithm but

- not goal directed → use 'magic sets':
  program and query can be transformed into a new program such that for the new program, forward chaining is goal directed with respect to the query

- semi naive algorithm throws away information about partial instantiations like \( h(a, Y) \leftarrow f(a), g(Y) \)
  → this information could instead be stored to reduce the number of required computations

For advanced settings like a database that changes during evaluation there are specialized methods like 'Incremental Query Evaluation using Conjunctive Queries'
Thank you