Overview

Epilog is a theorem prover for Relational Logic. It is sound and complete. It is at least as efficient as Model Elimination, and it is arguably more efficient. It is somewhat more intuitive than ordinary Resolution.

Features:
- Rule Form instead of Clausal Form
- Model Elimination
- Iterative Deepening
- Caching
Rule Form

Rule Form is the same as clausal form except that clauses are expressed using “rule” syntax rather than set notation.

There are two cases.
Premise form is used for premises.
Question form is used for desired conclusions.

Premises

Premises are expressed as rules.

\[
\begin{align*}
\langle p \rangle & \quad p \Leftarrow \quad p : - \\
\langle \neg p \rangle & \quad \neg p \Leftarrow \quad \neg p : - \\
\langle r, \neg p, \neg q \rangle & \quad r \Leftarrow p \land q \quad r : \neg p, q
\end{align*}
\]

NB: \( \langle \psi, \neg \varphi_1, \ldots, \neg \varphi_n \rangle \) is equivalent to \( \psi \Leftarrow \varphi_1 \land \ldots \land \varphi_n \).
Conclusions

Conclusions are expressed as *questions*.

\[
\langle p \rangle \quad \neg p? \\
\langle \neg p \rangle \quad p? \\
\langle \neg p, \neg q, r \rangle \quad p \land q \land \neg r?
\]

Note that \(\langle \neg \phi_1, \ldots, \neg \phi_n \rangle\) is equivalent to \((\neg \phi_1 \lor \ldots \lor \neg \phi_n)\).

Note that \((\neg \phi_1 \lor \ldots \lor \neg \phi_n)\) is equivalent to \(\neg (\phi_1 \land \ldots \land \phi_n)\).

Backward Chaining

Backward Chaining is the same as reduction except that it works on rule form rather than clausal form.

\[
\phi \iff \neg \psi_1 \land \ldots \land \neg \psi_m \\
\psi \land \psi_1 \land \ldots \land \psi_n? \\
\begin{array}{c}
\phi_1 \land \ldots \land \neg \psi_m \land \neg [\psi_1 \land \ldots \land \psi_n]? \sigma
\end{array}
\]

where \(\sigma = mgu(\phi, \psi)\)

Reduced literals need be retained only for non-Horn premises.

Cancellation and Dropping are analogous.
Example

1. \( m \Leftarrow \) Premise
2. \( p \Leftarrow m \) Premise
3. \( q \Leftarrow m \) Premise
4. \( r \Leftarrow p \land q \) Premise
5. \( r \) Goal
6. \( p \land q \) 4,5
7. \( m \land q \) 2,6
8. \( q \) 1,7
9. \( m \) 3,8
10. \( ? \) 1,9

Is Art the Grandparent of Coe?

1. \( p(art,bob) \) Premise
2. \( p(art,bud) \) Premise
3. \( p(bob,cal) \) Premise
4. \( p(bud,coe) \) Premise
5. \( g(x,z) \Leftarrow p(x,y) \land p(y,z) \) Premise
6. \( g(art,z) \)? Goal
7. \( p(art,y) \land p(y,z) \) 5,6
8. \( p(bud,z) \) 2,7
9. \( \bot \) 4,8
Non-Horn Example

\{p \lor q, \ p \lor \neg q, \ \neg p \lor q\}

\models

p \land q

Non-Horn Example

1. \ p \iff \neg q \quad p \lor q

2. \ q \iff \neg p \quad p \lor q

3. \ p \iff q \quad p \lor \neg q

4. \ \neg q \iff \neg p \quad p \lor \neg q

5. \ \neg p \iff \neg q \quad \neg p \lor q

6. \ q \iff p \quad \neg p \lor q

7. \ \neg p \iff q \quad \neg (p \land q)

8. \ \neg q \iff p \quad \neg (p \land q)

9. \ p \land q?

10. \ \neg q \land [p] \land q? \quad 1, 9

11. \ \neg p \land [\neg q] \land [p] \land q? \quad 4, 10

12. \ [\neg q] \land [p] \land q? \quad 11

13. \ [p] \land q? \quad 12

14. \ q? \quad 13

15. \ p \land [q]? \quad 6, 14

16. \ \neg q \land [p] \land [q]? \quad 1, 15

17. \ [p] \land [q]? \quad 16

18. \ [q]? \quad 17

19. \ ? \quad 18

Goal
Answer Extraction

To extract answers start with definition of goal relation rather than question.

\[ p(x,y) \land q(y,z) \]

\[ goal(x,z) \iff p(x,y) \land q(y,z) \]

Answer Extraction Rule

\[ \varphi \iff \varphi_1 \land \ldots \land \varphi_m \]
\[ \gamma \iff \psi \land \psi_1 \land \ldots \land \psi_n \]

\[ (\gamma \iff \varphi_1 \land \ldots \land \varphi_m \land [\psi] \land \psi_1 \land \ldots \land \psi_n)_{\sigma} \]

where \( \sigma = \text{mgu}(\varphi, \psi) \)

where \( \gamma \) is a goal literal
Example

1. \( m(a,a) \) \quad \text{Premise}
2. \( p(x,y) \iff m(x,y) \) \quad \text{Premise}
3. \( q(x,y) \iff m(x,y) \) \quad \text{Premise}
4. \( r(x,z) \iff p(x,y) \land q(y,z) \) \quad \text{Premise}
5. \( \text{goal}(x,z) \iff r(x,z) \) \quad \text{Goal}
6. \( \text{goal}(x,z) \iff p(x,y) \land q(y,z) \) \quad 4, 5
7. \( \text{goal}(x,z) \iff m(x,y) \land q(y,z) \) \quad 2, 6
8. \( \text{goal}(a,z) \iff q(a,z) \) \quad 1, 7
9. \( \text{goal}(a,z) \iff m(a,z) \) \quad 3, 8
10. \( \text{goal}(a,a) \iff \) \quad 1, 9

Who Are the Grandchildren of Art?

1. \( p(art,bob) \) \quad \text{Premise}
2. \( p(art,bud) \) \quad \text{Premise}
3. \( p(bob,cal) \) \quad \text{Premise}
4. \( p(bud,coe) \) \quad \text{Premise}
5. \( g(x,z) : \neg p(x,y), p(y,z) \) \quad \text{Premise}
6. \( \text{goal}(z) : \neg g(art,z) \) \quad \text{Goal}
7. \( \text{goal}(z) : \neg p(x,y), p(y,z) \) \quad 5, 6
8. \( \text{goal}(z) : \neg p(bob,z) \) \quad 1, 7
9. \( \text{goal}(cal) \) \quad 3, 8
10. \( \text{goal}(art) : \neg p(bud,z) \) \quad 2, 7
11. \( \text{goal}(coe) \) \quad 4, 10
Troubling Example

1. \( p(a) \iff \neg p(b) \quad p(a) \lor p(b) \)
2. \( p(b) \iff \neg p(a) \quad p(a) \lor p(b) \)
3. \( \text{goal}(x) \iff p(x) \quad \text{Goal} \)
4. \( \neg p(x) \iff \neg \text{goal}(x) \quad \text{Goal} \)
5. \( \text{goal}(a) \iff \neg p(b) \land [p(a)] \quad 1,3 \)
6. \( \text{goal}(a) \iff \neg \text{goal}(b) \land [\neg p(b)] \land [p(a)] \quad 4,5 \)

Multiple Goal Rule

\[ \gamma \iff \neg \gamma' \land \psi \land \psi_1 \land \ldots \land \psi_n \]
\[ \gamma \lor \gamma' \iff \psi \land \psi_1 \land \ldots \land \psi_n \]
where \( \gamma \) and \( \gamma' \) are goal literals
Example

1. \( p(a) \iff \neg p(b) \quad p(a) \lor p(b) \)
2. \( p(b) \iff \neg p(a) \quad p(a) \lor p(b) \)
3. \( \text{goal}(x) \iff p(x) \ Quad \text{Goal} \)
4. \( \neg p(x) \iff \neg \text{goal}(x) \ Quad \text{Goal} \)
5. \( \text{goal}(a) \iff \neg p(b) \land [p(a)] \quad 1,3 \)
6. \( \text{goal}(a) \iff \neg \text{goal}(b) \land [\neg p(b)] \land [p(a)] \quad 4,5 \)
7. \( \text{goal}(a) \lor \text{goal}(b) \iff [\neg p(b)] \land [p(a)] \quad 6 \)
8. \( \text{goal}(a) \lor \text{goal}(b) \iff [p(a)] \quad 7 \)
9. \( \text{goal}(a) \lor \text{goal}(b) \iff 8 \)

Search Space

Question:

\[ a? \]

Premises

\begin{align*}
    a & \iff b \\
    b & \iff e \\
    c & \iff g \\
    d & \iff i \\
    e & \iff f \\
    f & \iff h \\
    g & \iff j \\
\end{align*}
Search Space

Given the Input Restriction, the search space looks like a simple tree or graph.

There are different schedules for searching the tree or graph.

Breadth First Search

Advantage: Finds shortest path
Disadvantage: Saves lots of intermediate information
Depth First Search

\[ a \ b \ e \ f \ c \ g \ h \ d \ i \ j \]

Advantage: Small intermediate storage
Disadvantage: Susceptible to garden paths
Disadvantage: Susceptible to infinite loops

Time Comparison

Analysis for branching 2 and depth \( d \) and solution at depth \( k \)

\[
\begin{array}{ccc}
\text{Time} & \text{Best} & \text{Worst} \\
\hline
\text{Depth} & k & 2^d - 2^{d-k} \\
\text{Breadth} & 2^{k-1} & 2^k - 1 \\
\end{array}
\]
Time Comparison

Analysis for branching $b$ and depth $d$ and solution at depth $k$.

<table>
<thead>
<tr>
<th>Time</th>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>$k$</td>
<td>$\frac{b^d - b^{d-k}}{b-1}$</td>
</tr>
<tr>
<td>Breadth</td>
<td>$\frac{b^k - 1}{b-1} + 1$</td>
<td>$\frac{b^k - 1}{b-1}$</td>
</tr>
</tbody>
</table>

Space Comparison

Worst Case Space Analysis for search depth $d$ and depth $k$.

<table>
<thead>
<tr>
<th>Space</th>
<th>Binary</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>$d$</td>
<td>$(b-1) \times (d-1) + 1$</td>
</tr>
<tr>
<td>Breadth</td>
<td>$2^{k-1}$</td>
<td>$b^{k-1}$</td>
</tr>
</tbody>
</table>
Iterative Deepening

Run depth-limited search repeatedly,
starting with a small initial depth,
incrementing on each iteration,
until success or run out of alternatives.

Example

```
a
a b c d
a b e f c g h d i j
```

Advantage: Small intermediate storage
Advantage: Finds shortest path
Advantage: Not susceptible to garden paths
Advantage: Not susceptible to infinite loops
Time Comparison

<table>
<thead>
<tr>
<th>Depth</th>
<th>Iterative</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>57</td>
<td>31</td>
</tr>
<tr>
<td>(n)</td>
<td>(2^{n+1} - n - 2)</td>
<td>(2^n - 1)</td>
</tr>
</tbody>
</table>

General Result

Theorem: The cost of iterative deepening search is \(b/(b-1)\) times the cost of depth-first search (where \(b\) is the branching factor).

See Korf.
Caching Motivation

\[
\begin{align*}
\text{fib}(0) &= 1 \\
\text{fib}(1) &= 1 \\
\text{fib}(x) &= \text{fib}(x-1) + \text{fib}(x-2)
\end{align*}
\]

\[
\begin{align*}
\text{fib}(4) &= 5 & \text{fib}(4) &= 5 \\
\text{fib}(3) &= 3 & \text{fib}(3) &= 3 \\
\text{fib}(2) &= 2 & \text{fib}(2) &= 2 \\
\text{fib}(1) &= 1 & \text{fib}(1) &= 1 \\
\text{fib}(0) &= 1 & \text{fib}(0) &= 1
\end{align*}
\]

Especially important in ID, where work is repeated.

Partial Caching

Idea - when derive atomic result, add result to database for use in future deductions.

If same goal appears again, can find result quickly.

If not, may lead to more work.
Example Without Caching

\[
\begin{align*}
f(a,b) \\
f(a,c) \\
f(a,d) \\
m(d,e) \\
p(X,Z) & :- m(X,Y) \\
p(X,Z) & :- f(X,Y) \\
g(X,Z) & :- p(X,Y) & \& p(Y,Z) \\
g(a,e)? \\
p(a,Y) & \& p(Y,e)? \\
\text{branching factor 2}
\end{align*}
\]

Problem With Partial Caching

\[
\begin{align*}
f(a,b) \\
f(a,c) \\
f(a,d) \\
m(d,e) \\
p(X,Z) & :- m(X,Y) \\
p(X,Z) & :- f(X,Y) \\
g(X,Z) & :- p(X,Y) & \& p(Y,Z) \\
p(a,c) \\
p(a,d) \\
g(a,e)? \\
p(a,Y) & \& p(Y,e)? \\
\text{branching factor 4!!}
\end{align*}
\]
Complete Caching

Caching results in general resolution / model elimination / chaining can save work but also increases the search space.

If the search is done in DF fashion, it is possible to cache failures as well as successes (complete caching) because all possible derivations are done before exit. If failure, can ignore rules if subgoal occurs again.

With this approach one either uses the cache and only the cache (if complete) or uses rules (if incomplete). In this way, one gets the benefits of caching and one never does any search more than once.

Complete Caching With ID

What happens with iterative deepening? System may not try all derivations before failing due to depth limit. Searching deeper might lead to a solution. So we still need to examine rules.

Solution: Annotate cache with depth limit. Can avoid re-using rules so long as depth limit not increased.

Astrachan and Stickel - Horn clauses
Don Geddis - Non-Horn case
Astrachan and Stickel

goal pair is \(<q, n>\) where \(n\) is maximum depth to be used.

\(<\text{anc}(x, y), 3>\>

Cache store contains cache solutions, initially empty.
Cache solution is \(<q, n>\) where \(n\) is min depth where \(q\) found.

\(<\text{anc}(a, b), 2>\>

Cache directory contains cache templates, initially empty.
Cache template is \(<q, m, s>\) where \(m\) is the depth searched.
If \(s \leq m\), then \(s\) is smallest depth needed to solve \(q\).
Otherwise, there is no solution at any depth \(\leq m\).

\(<\text{anc}(x, y), 3, 2>\>

Cache is complete for \(<q, n>\) iff includes all solutions to depth \(n\).

---

Astrachan and Stickel (continued)

\(solve(q, n)\) - do the following before the usual processing:
Find template \(<\psi, m, s>\>
check \(q = \psi\) or subsumed (depending on implementation)

If \(m = n\) (complete)
then if \(n \geq s\) then solve with cache else return false

Otherwise do inference
when procedure finds solution \(q\) at depth \(n\)
cache store gets \(<q, n>\) and greater depth solns removed.
any matching cache template \(<q, m, s>\) updated if \(n < s\).

when procedure fails after searching to depth \(n\) where \(n > m\)
cache template is updated \(<q, m, s> \rightarrow <q, n, s>\)
Other Benefits of DF / ID Search

Question literals, reduced literals, and goal literals can be kept on stacks.

Variables do not need to be plugged in along the way; instead, they can be kept in binding lists (possibly stack-allocated).

Upshot - less computation cost (since no copying) and less garbage collection (since no copying).

Ease of Understanding of Proof Process

Call: \( r \)?
1. \( m \Leftarrow \) Premise

Call: \( p \)?
2. \( p \Leftarrow m \) Premise

Call: \( m \)?
3. \( q \Leftarrow m \) Premise

Exit: \( m \)
4. \( r \Leftarrow p \land q \) Premise

Exit: \( p \)
5. \( r \) Goal

Call: \( q \)?
6. \( p \land q ? \) 4,5

Call: \( m \)?
7. \( m \land q ? \) 2,6

Exit: \( m \)

Exit: \( q \)
8. \( q \) 1,7

Exit: \( r \)
9. \( m ? \) 3,8

10. ? 1,9

Depth-First Search supports tracing and debugging.
Highly important on problems with many rules.
Ease of Understanding of Proof Process

Call: \( r(x) \)?
1. \( p(a) \leftarrow \)

Call: \( p(x) \)?
2. \( p(b) \leftarrow \)

Exit: \( p(a) \)
3. \( q(b) \leftarrow \)

Call: \( q(a) \)?
4. \( r(x) \leftarrow p(x) \land q(x) \)

Fail: \( q(a) \)?
5. \( r(x) \leftarrow \)

Redo: \( p(x) \)?

Exit: \( p(b) \)

Call: \( q(b) \)?

Exit: \( q(b) \)?

Exit: \( r(b) \)

Related Systems

Prolog (David Warren)
- Horn Clauses only
- Depth-First Search has potential for infinite loops
- No occur check in unifier

Prolog Technology Theorem Prover (Mark Stickel)
- essentially the same as Epilog

Epilog (Stanford University)
- essentially the same as PTTP