

Epilog

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
- Backward Chaining variant of the ME rule
- Iterative Deepening rather than Breadth-First Search

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.

3

Premises

Premises are expressed as *rules*.

$\langle p \rangle$	$p \Leftarrow$	$p :-$
$\langle \neg p \rangle$	$\neg p \Leftarrow$	$\neg p :-$
$\langle r, \neg p, \neg q \rangle$	$r \Leftarrow p \wedge q$	$r :- p, q$

NB: $\langle \psi, \neg \varphi_1, \dots, \neg \varphi_n \rangle$ is equivalent to $(\psi \Leftarrow \varphi_1 \wedge \dots \wedge \varphi_n)$.

4

Conclusions

Conclusions are expressed as *questions*.

$$\begin{array}{ll} \langle p \rangle & \neg p? \\ \langle \neg p \rangle & p? \\ \langle \neg p, \neg q, r \rangle & p \wedge q \wedge \neg r? \end{array}$$

Note that $\langle \neg\varphi_1, \dots, \neg\varphi_n \rangle$ is equivalent to $(\neg\varphi_1 \vee \dots \vee \neg\varphi_n)$.
 Note that $(\neg\varphi_1 \vee \dots \vee \neg\varphi_n)$ is equivalent to $\neg(\varphi_1 \wedge \dots \wedge \varphi_n)$.

5

Backward Chaining

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

$$\frac{\varphi \Leftarrow \varphi_1 \wedge \dots \wedge \varphi_m \quad \psi \wedge \psi_1 \wedge \dots \wedge \psi_n?}{\varphi_1 \wedge \dots \wedge \varphi_m \wedge [\psi] \wedge \psi_1 \wedge \dots \wedge \psi_n? \sigma}$$

where $\sigma = mgu(\varphi, \psi)$

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

Cancellation and Dropping are analogous.

6

Example

- | | | |
|-----|---------------------------|---------|
| 1. | $m \Leftarrow$ | Premise |
| 2. | $p \Leftarrow m$ | Premise |
| 3. | $q \Leftarrow m$ | Premise |
| 4. | $r \Leftarrow p \wedge q$ | Premise |
| 5. | $r?$ | Goal |
| 6. | $p \wedge q?$ | 4,5 |
| 7. | $m \wedge q?$ | 2,6 |
| 8. | $q?$ | 1,7 |
| 9. | $m?$ | 3,8 |
| 10. | $?$ | 1,9 |

7

Is Art the Grandparent of Coe?

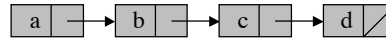
- | | | |
|----|---|---------|
| 1. | $p(\text{art}, \text{bob})$ | Premise |
| 2. | $p(\text{art}, \text{bud})$ | Premise |
| 3. | $p(\text{bob}, \text{cal})$ | Premise |
| 4. | $p(\text{bud}, \text{coe})$ | Premise |
| 5. | $g(x, z) \Leftarrow p(x, y) \wedge p(y, z)$ | Premise |
| 6. | $g(\text{art}, z)?$ | Goal |
| 7. | $p(\text{art}, y) \wedge p(y, z)?$ | 5,6 |
| 8. | $p(\text{bud}, z)?$ | 2,7 |
| 9. | \perp | 4,8 |

8

Variable Length Lists

Example

$[a,b,c,d]$



Representation as Term

$cons(a, cons(b, cons(c, cons(d, nil))))$

Shorthand

$(a . (b . (c . (d . nil))))$

Shorthand

$[a,b,c,d]$

9

List Membership Example

1. $member(x, nil)$ Premise
2. $member(x, (y . z)) \Leftarrow member(x, z)$ Premise
3. $member(c, (a . (b . (c . nil))))?$ Goal
4. $member(c, (b . (c . nil)))?$ 2, 3
5. $member(c, (c . nil))?$ 2, 4
6. ? 1, 5

10

Non-Horn Example

$$\begin{aligned}
 & p \vee q \\
 & p \vee \neg q \\
 & \neg p \vee q \\
 & \neg p \vee \neg q
 \end{aligned}$$

11

Non-Horn Example

1.	$p \Leftarrow \neg q$	$p \vee q$		9.	$p \wedge q?$	Goal
2.	$q \Leftarrow \neg p$	$p \vee q$		10.	$\neg q \wedge [p] \wedge q?$	1,9
3.	$p \Leftarrow q$	$p \vee \neg q$		11.	$\neg p \wedge [\neg q] \wedge [p] \wedge q?$	4,10
4.	$\neg q \Leftarrow \neg p$	$p \vee \neg q$		12.	$[\neg q] \wedge [p] \wedge q?$	11
5.	$\neg p \Leftarrow \neg q$	$\neg p \vee q$		13.	$[p] \wedge q?$	12
6.	$q \Leftarrow p$	$\neg p \vee q$		14.	$q?$	13
7.	$\neg p \Leftarrow q$	$\neg(p \wedge q)$		15.	$p \wedge [q]?$	6,14
8.	$\neg q \Leftarrow p$	$\neg(p \wedge q)$		16.	$\neg q \wedge [p] \wedge [q]?$	1,15
				17.	$[p] \wedge [q]?$	16
				18.	$[q]?$	17
				19.	$?$	18

12

Answer Extraction

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

$$p(x,y) \wedge q(y,z)?$$

$$goal(x,z) \Leftarrow p(x,y) \wedge q(y,z)$$

13

Answer Extraction Rule

$$\varphi \Leftarrow \varphi_1 \wedge \dots \wedge \varphi_m$$

$$\gamma \Leftarrow \psi \wedge \psi_1 \wedge \dots \wedge \psi_n$$

$$\frac{}{(\gamma \Leftarrow \varphi_1 \wedge \dots \wedge \varphi_m \wedge [\psi] \wedge \psi_1 \wedge \dots \wedge \psi_n) \sigma}$$

where $\sigma = mgu(\varphi, \psi)$

where γ is a goal literal

14

Example

- | | | |
|-----|--|---------|
| 1. | $m(a, a) \Leftarrow$ | Premise |
| 2. | $p(x, y) \Leftarrow m(x, y)$ | Premise |
| 3. | $q(x, y) \Leftarrow m(x, y)$ | Premise |
| 4. | $r(x, z) \Leftarrow p(x, y) \wedge q(y, z)$ | Premise |
| 5. | $goal(x, z) \Leftarrow r(x, z)$ | Goal |
| 6. | $goal(x, z) \Leftarrow p(x, y) \wedge q(y, z)$ | 4, 5 |
| 7. | $goal(x, z) \Leftarrow m(x, y) \wedge q(y, z)$ | 2, 6 |
| 8. | $goal(a, z) \Leftarrow q(a, z)$ | 1, 7 |
| 9. | $goal(a, z) \Leftarrow m(a, z)$ | 3, 8 |
| 10. | $goal(a, a) \Leftarrow$ | 1, 9 |

15

Who Are the Grandchildren of Art?

- | | | |
|-----|-----------------------------------|---------|
| 1. | $p(art, bob)$ | Premise |
| 2. | $p(art, bud)$ | Premise |
| 3. | $p(bob, cal)$ | Premise |
| 4. | $p(bud, coe)$ | Premise |
| 5. | $g(x, z) : \neg p(x, y), p(y, z)$ | Premise |
| 6. | $goal(z) : \neg g(art, z)$ | Goal |
| 7. | $goal(z) : \neg p(x, y), p(y, z)$ | 5, 6 |
| 8. | $goal(z) : \neg p(bob, z)$ | 1, 7 |
| 9. | $goal(cal)$ | 3, 8 |
| 10. | $goal(art) : \neg p(bud, z)$ | 2, 7 |
| 11. | $goal(coe)$ | 4, 10 |

16

Troubling Example

- | | | |
|----|--|------------------|
| 1. | $p(a) \Leftarrow \neg p(b)$ | $p(a) \vee p(b)$ |
| 2. | $p(b) \Leftarrow \neg p(a)$ | $p(a) \vee p(b)$ |
| 3. | $goal(x) \Leftarrow p(x)$ | Goal |
| 4. | $\neg p(x) \Leftarrow \neg goal(x)$ | Goal |
| 5. | $goal(a) \Leftarrow \neg p(b) \wedge [p(a)]$ | 1, 3 |
| 6. | $goal(a) \Leftarrow \neg goal(b) \wedge [\neg p(b)] \wedge [p(a)]$ | 4, 5 |

17

Multiple Goal Rule

$$\frac{\gamma \Leftarrow \neg \gamma' \wedge \psi \wedge \psi_1 \wedge \dots \wedge \psi_n}{\gamma \vee \gamma' \Leftarrow \psi \wedge \psi_1 \wedge \dots \wedge \psi_n}$$

where γ and γ' are goal literals

18

Example

1. $p(a) \Leftarrow \neg p(b)$	$p(a) \vee p(b)$
2. $p(b) \Leftarrow \neg p(a)$	$p(a) \vee p(b)$
3. $goal(x) \Leftarrow p(x)$	Goal
4. $\neg p(x) \Leftarrow \neg goal(x)$	Goal
5. $goal(a) \Leftarrow \neg p(b) \wedge [p(a)]$	1, 3
6. $goal(a) \Leftarrow \neg goal(b) \wedge [\neg p(b)] \wedge [p(a)]$	4, 5
7. $goal(a) \vee goal(b) \Leftarrow [\neg p(b)] \wedge [p(a)]$	6
8. $goal(a) \vee goal(b) \Leftarrow [p(a)]$	7
9. $goal(a) \vee goal(b) \Leftarrow$	8

19

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.	$r \Leftarrow p \wedge q$	Premise
Exit: p	5.	$r?$	Goal
Call: $q?$	6.	$p \wedge q?$	4, 5
Call: $m?$	7.	$m \wedge q?$	2, 6
Exit: m	8.	$q?$	1, 7
Exit: q	9.	$m?$	3, 8
Exit: r	10.	?	1, 9

Depth-First Search supports tracing and debugging.
Highly important on problems with many rules.

20

Ease of Understanding of Proof Process

Call: $r(x)?$

Call: $p(x)?$

Exit: $p(a)$

Call: $q(a)?$

Fail: $q(a)?$

Redo: $p(x)?$

Exit: $p(b)$

Call: $q(b)?$

Exit: $q(b)?$

Exit: $r(b)$

1. $p(a) \Leftarrow$

2. $p(b) \Leftarrow$

3. $q(b) \Leftarrow$

4. $r(x) \Leftarrow p(x) \wedge q(x)$

5. $r(x)?$

21

Search Space

Question:

$a?$

Premises

$a \Leftarrow b$

$a \Leftarrow c$

$a \Leftarrow d$

$b \Leftarrow e$

$c \Leftarrow g$

$d \Leftarrow i$

$b \Leftarrow f$

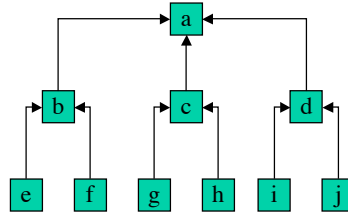
$c \Leftarrow h$

$d \Leftarrow j$

22

Search Space

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

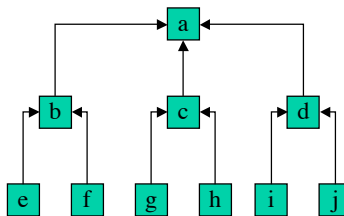


There are different schedules for searching the tree or graph.

23

Breadth First Search

a b c d e f g h i j



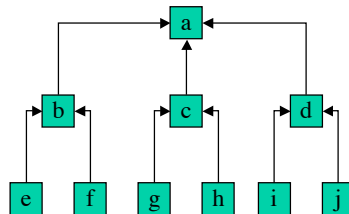
Advantage: Finds shortest path

Disadvantage: Saves lots of intermediate information

24

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

25

Time Comparison

Analysis for branching 2 and depth d and solution at depth k

<i>Time</i>	<i>Best</i>	<i>Worst</i>
<i>Depth</i>	k	$2^d - 2^{d-k}$
<i>Breadth</i>	2^{k-1}	$2^k - 1$

26

Time Comparison

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

<i>Time</i>	<i>Best</i>	<i>Worst</i>
<i>Depth</i>	k	$\frac{b^d - b^{d-k}}{b-1}$
<i>Breadth</i>	$\frac{b^{k-1} - 1}{b-1} + 1$	$\frac{b^k - 1}{b-1}$

27

Space Comparison

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

<i>Space</i>	<i>Binary</i>	<i>General</i>
<i>Depth</i>	d	$(b-1) \times (d-1) + 1$
<i>Breadth</i>	2^{k-1}	b^{k-1}

28

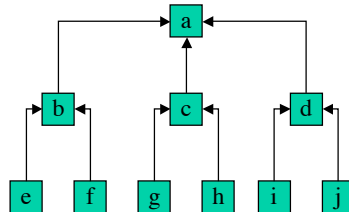
Iterative Deepening

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

29

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

30

Time Comparison

<i>Depth</i>	<i>Iterative</i>	<i>Depth</i>
1	1	1
2	4	3
3	11	7
4	26	15
5	57	31
n	$2^{n+1} - n - 2$	$2^n - 1$

31

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.

32

Caching Motivation

fib(0)=1
fib(1)=1
fib(x)=fib(x-1) + fib(x-2)

fib(4)
fib(3)
fib(2)
fib(1)
fib(0)
fib(1)
fib(2)
fib(1)
fib(0)

33

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 / ID fashion, it is possible to cache both results *and* failures so that one gets the benefits of caching and one *never* does any search more than once.

34

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).

Results and failures can be cached and re-used to save redundant computation without increasing the size of the search space.

35

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 (yours truly)

essentially the same as PTPP

36