Outline

1. Logic in Computer Science
   - Horn clauses

2. Prolog execution model
   - Unification and backtracking
   - A Meta-interpreter

3. References
Outline for section 1

1. Logic in Computer Science
   - Horn clauses

2. Prolog execution model
   - Unification and backtracking
   - A Meta-interpreter

3. References
Thor is a parent of Magni.

Thor is a parent of Modi.

Siblings have a common parent.

∀ x ∀ y (∃ p. Parent(p, x) ∧ Parent(p, y) → Sibling(x, y))

Q: Who is the parent of Modi?

A: Thor. (Satisfiable with X = Thor)

Q: Are Magni and Modi siblings?

A: Yes. (Tautology)
First-order Logic: Example

Thor is a parent of Magni.  \( \text{Parent}(\text{THOR}, \text{MAGNI}) \)
First-order Logic: Example

Thor is a parent of Magni.  
Thor is a parent of Modi.  

$\text{Parent}(\text{Thor, Magni})$

Q: Who is the parent of Modi?  
A: Thor.  
(Satisfiable with $X = \text{Thor}$)

Q: Are Magni and Modi siblings?  
A: Yes.  
(Tautology)
Thor is a parent of Magni.  \( \text{Parent}(\text{THOR}, \text{MAGNI}) \)
Thor is a parent of Modi.  \( \text{Parent}(\text{THOR}, \text{MODI}) \)

Siblings have a common parent.
\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

Q: Who is the parent of Modi?
A: Thor. (Satisfiable with \( X = \text{THOR} \))

Q: Are Magni and Modi siblings?
A: Yes. (Tautology)
Thor is a parent of Magni. \( \text{Parent}(\text{Thor, Magni}) \)
Thor is a parent of Modi. \( \text{Parent}(\text{Thor, Modi}) \)
Siblings have a common parent.
First-order Logic: Example

Thor is a parent of Magni.
Thor is a parent of Modi.
Siblings have a common parent.

\[
\text{Parent}(\text{THOR, MAGNI}) \\
\text{Parent}(\text{THOR, MODI}) \\
\forall x \forall y. (\exists \ p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y)
\]
First-order Logic: Example

Thor is a parent of Magni.  \( Parent(\text{THOR, MAGNI}) \)
Thor is a parent of Modi.  \( Parent(\text{THOR, MODI}) \)
Siblings have a common parent.  \( \forall x \forall y. (\exists p. Parent(p, x) \land Parent(p, y)) \rightarrow Sibling(x, y) \)

Q: Who is the parent of Modi?
A: Thor. (Satisfiable with \( X = \text{THOR} \))

Q: Are Magni and Modi siblings?
A: Yes. (Tautology)
Thor is a parent of Magni.  \( \text{Parent}(\text{Thor, Magni}) \)

Thor is a parent of Modi.  \( \text{Parent}(\text{Thor, Modi}) \)

Siblings have a common parent.  \( \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \)

Q: Who is the parent of Modi?  \( \text{Parent}(\text{X, Modi}) \)
First-order Logic: Example

Thor is a parent of Magni.  
Thor is a parent of Modi. 
Siblings have a common parent.  

Q: Who is the parent of Modi?  
A: Thor.

$\forall x \forall y. (\exists p. Parent(p, x) \land \exists p. Parent(p, y) \rightarrow Sibling(x, y)$

$Parent(\text{THOR}, \text{MAGNI})$

$Parent(\text{THOR}, \text{MODI})$

$Parent(X, \text{MODI})$
Thor is a parent of Magni.  
\[ \text{Parent}(\text{Thor, Magni}) \]

Thor is a parent of Modi.  
\[ \text{Parent}(\text{Thor, Modi}) \]

Siblings have a common parent.  
\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

Q: Who is the parent of Modi?  
A: Thor.  
\[ \text{Parent}(\text{X, Modi}) \]
(Satisfiable with \( X = \text{Thor} \))
First-order Logic: Example

Thor is a parent of Magni.
Thor is a parent of Modi.
Siblings have a common parent.

\[ \forall x \forall y. (\exists p. Parent(p, x) \land Parent(p, y)) \rightarrow Sibling(x, y) \]

Q: Who is the parent of Modi?
A: Thor.

Q: Are Magni and Modi siblings?

\[ Parent(X, MODI) \]
(Satisfiable with \( X = \text{THOR} \))
First-order Logic: Example

Thor is a parent of Magni.  
Thor is a parent of Modi.  
Siblings have a common parent.

\[
\forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y)
\]

Q: Who is the parent of Modi?  
A: Thor.  
(Satisfiable with \( X = \text{Thor} \))

Q: Are Magni and Modi siblings?  
A: Yes.  
(Satisfiable with Magni and Modi)
Thor is a parent of Magni.
Thor is a parent of Modi.
Siblings have a common parent.

Q: Who is the parent of Modi?
A: Thor.

Q: Are Magni and Modi siblings?
A: Yes.
First-order Logic: Example

Thor is a parent of Magni.  \(\text{Parent}(\text{Thor}, \text{Magni})\)
Thor is a parent of Modi.  \(\text{Parent}(\text{Thor}, \text{Modi})\)
Siblings have a common parent.  \(\forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y)\)

Q: Who is the parent of Modi?
A: Thor.

Q: Are Magni and Modi siblings?
A: Yes.

\(\text{Parent}(X, \text{Modi})\)
(Satisfiable with \(X = \text{Thor}\))

\(\text{Sibling}(\text{Magni}, \text{Modi})\)
(Tautology)
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. Parent(p, x) \land Parent(p, y)) \rightarrow Sibling(x, y) \]

\[ \forall x \forall y. Sibling(x, y) \leftarrow (\exists p. Parent(p, x) \land Parent(p, y)) \]
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \leftarrow (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \neg (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
A horn clause is a disjunction of literals with at most one non-negated literal.

\[
\forall x \forall y. (\exists p. Parent(p, x) \land Parent(p, y)) \rightarrow Sibling(x, y)
\]

\[
\forall x \forall y. Sibling(x, y) \leftarrow (\exists p. Parent(p, x) \land Parent(p, y))
\]

\[
\forall x \forall y. Sibling(x, y) \lor \neg (\exists p. Parent(p, x) \land Parent(p, y))
\]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \leftarrow (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \neg (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \forall p. \neg (\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg (\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[
\forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y)
\]

\[
\forall x \forall y. \text{Sibling}(x, y) \leftarrow (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y))
\]

\[
\forall x \forall y. \text{Sibling}(x, y) \lor \neg(\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y))
\]

\[
\forall x \forall y. \text{Sibling}(x, y) \lor \forall p. \neg(\text{Parent}(p, x) \land \text{Parent}(p, y))
\]

\[
\forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg(\text{Parent}(p, x) \land \text{Parent}(p, y))
\]

\[
\forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg\text{Parent}(p, x) \lor \neg\text{Parent}(p, y)
\]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \leftarrow (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \neg(\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \forall p. \neg(\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg(\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg \text{Parent}(p, x) \lor \neg \text{Parent}(p, y) \]

\[ [\text{Sibling}(x, y), \neg \text{Parent}(p, x), \neg \text{Parent}(p, y)] \]
Horn clauses

A **horn clause** is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \leftarrow (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \neg(\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y. \text{Sibling}(x, y) \lor \forall p. \neg(\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg(\text{Parent}(p, x) \land \text{Parent}(p, y)) \]

\[ \forall x \forall y \forall p. \text{Sibling}(x, y) \lor \neg \text{Parent}(p, x) \lor \neg \text{Parent}(p, y) \]

\[ [\text{Sibling}(x, y), \neg \text{Parent}(p, x), \neg \text{Parent}(p, y)] \]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
Outline for section 2

1. Logic in Computer Science
   - Horn clauses

2. Prolog execution model
   - Unification and backtracking
   - A Meta-interpreter

3. References
(1) Thor is a parent of Magni.  
(2) Thor is a parent of Modi. 
(3) Siblings have a common parent.

\[ \text{Par(THOR, MAGNI)} \]

\[ \text{Par(THOR, MODI)} \]

(3) Siblings have a common parent.
(1) Thor is a parent of Magni. [\text{Par(THOR, MAGNI)}]
(2) Thor is a parent of Modi. [\text{Par(THOR, MODI)}]
(3) Siblings have a common parent.

\[ \forall x \forall y. (\exists p. \text{Par}(p, x) \land \text{Par}(p, y)) \rightarrow \text{Sib}(x, y) \]
(1) Thor is a parent of Magni. \[[\text{Par(THOR, MAGNI)}]\]
(2) Thor is a parent of Modi. \[[\text{Par(THOR, MODI)}]\]
(3) Siblings have a common parent.

\[
\forall x \forall y. (\exists p. \text{Par}(p, x) \land \text{Par}(p, y)) \rightarrow \text{Sib}(x, y)
\]

\[
\text{Sib}(x, y) \leftarrow \text{Par}(p, x) \land \text{Par}(p, y)
\]

\[
\text{sib}(X, Y) :- \text{par}(P, X), \text{par}(P, Y).
\]
(1) Thor is a parent of Magni. \[ Par(\text{THOR}, \text{MAGNI}) \]

(2) Thor is a parent of Modi. \[ Par(\text{THOR}, \text{MODI}) \]

(3) Siblings have a common parent.

\[ \forall x \forall y. (\exists p. Par(p, x) \land Par(p, y)) \rightarrow Sib(x, y) \]

\[ Sib(x, y) \leftarrow Par(p, x) \land Par(p, y) \]

\[ Sib(x, y) \lor \neg Par(p, x) \lor \neg Par(p, y) \]

\[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]
Unification and Resolution

(1) Thor is a parent of Magni.  [Par(THOR, MAGNI)]
(2) Thor is a parent of Modi.    [Par(THOR, MODI)]
(3) Siblings have a common parent.  [Sib(x, y), ¬Par(p, x), ¬Par(p, y)]

∀x∀y.(∃p.Par(p, x) ∧ Par(p, y)) → Sib(x, y)

Sib(x, y) ← Par(p, x) ∧ Par(p, y)         sib(X,Y) :- par(P,X), par(P,Y).
Sib(x, y) ∨ ¬Par(p, x) ∨ ¬Par(p, y)         [Sib(x, y), ¬Par(p, x), ¬Par(p, y)]
Unification and Resolution

(1) Thor is a parent of Magni. 
(2) Thor is a parent of Modi. 
(3) Siblings have a common parent. 
(4) Are Magni and Modi siblings?

\[ \text{Par}(\text{THOR}, \text{MAGNI}) \]
\[ \text{Par}(\text{THOR}, \text{MODI}) \]
\[ \text{Sib}(x, y), \neg \text{Par}(p, x), \neg \text{Par}(p, y) \]
(1) Thor is a parent of Magni. 
(2) Thor is a parent of Modi. 
(3) Siblings have a common parent. 
(4) Are Magni and Modi siblings?

\[ Par(\text{THOR}, \text{MAGNI}) \]
\[ Par(\text{THOR}, \text{MODI}) \]
\[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]
\[ \neg Sib(\text{MAGNI}, \text{MODI}) \]
(1) Thor is a parent of Magni. \[ \text{Par}(\text{Thor}, \text{Magni}) \]
(2) Thor is a parent of Modi. \[ \text{Par}(\text{Thor}, \text{Modi}) \]
(3) Siblings have a common parent. \[ \text{Sib}(x, y), \neg \text{Par}(p, x), \neg \text{Par}(p, y) \]
(4) Are Magni and Modi siblings? \[ \neg \text{Sib}(\text{Magni}, \text{Modi}) \]

Unify (4) and (3) with \( x = \text{Magni}, y = \text{Modi} \) and resolve...
(1) Thor is a parent of Magni. \[ Par(\text{THOR, MAGNI}) \]
(2) Thor is a parent of Modi. \[ Par(\text{THOR, MODI}) \]
(3) Siblings have a common parent. \[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]
(4) Are Magni and Modi siblings? \[ \neg Sib(\text{MAGNI, MODI}) \]

Unify (4) and (3) with \( x = \text{MAGNI}, y = \text{MODI} \) and resolve...

(5) \[ \neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI}) \]
Unification and Resolution

1. Thor is a parent of Magni.  
   \[ Par(\text{THOR}, \text{MAGNI}) \]

2. Thor is a parent of Modi.  
   \[ Par(\text{THOR}, \text{MODI}) \]

3. Siblings have a common parent.  
   \[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]

4. Are Magni and Modi siblings?  
   \[ \neg Sib(\text{MAGNI}, \text{MODI}) \]

Unify (4) and (3) with \( x = \text{MAGNI}, y = \text{MODI} \) and resolve...

5. \[ \neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI}) \]

Unify (5) and (1) with \( p = \text{THOR}, \text{MAGNI} = \text{MAGNI} \) and resolve...
(1) Thor is a parent of Magni. \[ Par(\text{THOR}, \text{MAGNI}) \]

(2) Thor is a parent of Modi. \[ Par(\text{THOR}, \text{MODI}) \]

(3) Siblings have a common parent. \[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]

(4) Are Magni and Modi siblings? \[ \neg Sib(\text{MAGNI}, \text{MODI}) \]

Unify (4) and (3) with \( x = \text{MAGNI}, y = \text{MODI} \) and resolve...

(5) \[ \neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI}) \]

Unify (5) and (1) with \( p = \text{THOR}, \text{MAGNI} = \text{MAGNI} \) and resolve...

(6) \[ \neg Par(\text{THOR}, \text{MODI}) \]
Unification and Resolution

(1) Thor is a parent of Magni. \[ Par(\text{THOR}, \text{MAGNI}) \]
(2) Thor is a parent of Modi. \[ Par(\text{THOR}, \text{MODI}) \]
(3) Siblings have a common parent. \[ Sib(x, y), \neg Par(p, x), \neg Par(p, y) \]
(4) Are Magni and Modi siblings? \[ \neg Sib(\text{MAGNI}, \text{MODI}) \]

Unify (4) and (3) with \( x = \text{MAGNI}, y = \text{MODI} \) and resolve... 

(5) \[ \neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI}) \]

Unify (5) and (1) with \( p = \text{THOR}, \text{MAGNI} = \text{MAGNI} \) and resolve...

(6) \[ \neg Par(\text{THOR}, \text{MODI}) \]

Unify (6) and (2) with \( \text{THOR} = \text{THOR}, \text{MODI} = \text{MODI} \) and resolve...
(1) Thor is a parent of Magni. \[Par(\text{THOR}, \text{MAGNI})]\n(2) Thor is a parent of Modi. \[Par(\text{THOR}, \text{MODI})]\n(3) Siblings have a common parent. \[Sib(x, y), \neg Par(p, x), \neg Par(p, y)\]\n(4) Are Magni and Modi siblings? \[\neg Sib(\text{MAGNI}, \text{MODI})\]

Unify (4) and (3) with \(x = \text{MAGNI}, y = \text{MODI}\) and resolve...

(5) \[\neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI})\]

Unify (5) and (1) with \(p = \text{THOR}, \text{MAGNI} = \text{MAGNI}\) and resolve...

(6) \[\neg Par(\text{THOR}, \text{MODI})\]

Unify (6) and (2) with \(\text{THOR} = \text{THOR}, \text{MODI} = \text{MODI}\) and resolve...

(7) [ ] Empty disjunction (contradiction)
Unification and Resolution

(1) Thor is a parent of Magni. \(\text{Par}(\text{THOR}, \text{MAGNI})\)
(2) Thor is a parent of Modi. \(\text{Par}(\text{THOR}, \text{MODI})\)
(3) Siblings have a common parent. \(\text{Sib}(x, y), \neg\text{Par}(p, x), \neg\text{Par}(p, y)\)
(4) Are Magni and Modi siblings? \(\neg\text{Sib}(\text{MAGNI}, \text{MODI})\)

Unify (4) and (3) with \(x = \text{MAGNI}, y = \text{MODI}\) and resolve...

(5) \(\neg\text{Par}(p, \text{MAGNI}), \neg\text{Par}(p, \text{MODI})\)

Unify (5) and (1) with \(p = \text{THOR}, \text{MAGNI} = \text{MAGNI}\) and resolve...

(6) \(\neg\text{Par}(\text{THOR}, \text{MODI})\)

Unify (6) and (2) with \(\text{THOR} = \text{THOR}, \text{MODI} = \text{MODI}\) and resolve...

(7) [ ] Empty disjunction (contradiction)

Negated query leads to contradiction, therefore the answer is yes.
parent(thor, magni).
parent(thor, modi).
sibling(X,Y) :- parent(P,X), parent(P,Y).

?- visible(+ all), trace, sibling(magni, modi).

Call : (7) sibling(magni, modi)
Unify : (7) sibling(magni, modi)
Call : (8) parent(X, magni)
Unify : (8) parent(thor, magni)
Exit : (8) parent(thor, magni)
Call : (8) parent(thor, modi)
Unify : (8) parent(thor, modi)
Exit : (8) parent(thor, modi)
Exit : (7) sibling(magni, modi) yes.
parent(thor, magni).
parent(thor, modi).
sibling(X,Y) :- parent(P,X), parent(P,Y).

?- visible(+all), trace, sibling(magni, modi).
parent(thor, magni).
parent(thor, modi).
sibling(X,Y) :- parent(P,X), parent(P,Y).

?- visible(+all), trace, sibling(magni, modi).

Call: (7) sibling(magni, modi)
Unify: (7) sibling(magni, modi)
   Call: (8) parent(X, magni)
   Unify: (8) parent(thor, magni)
   Exit: (8) parent(thor, magni)

   Call: (8) parent(thor, modi)
   Unify: (8) parent(thor, modi)
   Exit: (8) parent(thor, modi)
Exit: (7) sibling(magni, modi) yes.
Backtracking in Prolog

member(E, [E|_]).
member(E, [_|R]) :- member(E, R).

?- visible(+ all), trace, member(X, [1, 2]).
Backtracking in Prolog

\texttt{member(E,[E|_]).}
\texttt{member(E,[_|R]) :- member(E,R).}
\texttt{?- visible(+all), trace, member(X,[1,2]).}
Backtracking in Prolog

member(E,[E|_]).
member(E,[_|R]) :- member(E,R).
?- visible(+all), trace, member(X,[1,2]).

Call: (7) member(X, [1, 2])
Unify: (7) member(1, [1, 2])
Exit: (7) member(1, [1, 2]) X = 1 ;

Fail: (9) member(X, [])
Fail: (8) member(X, [2])
Fail: (7) member(X, [1, 2]) no.
Backtracking in Prolog

member(E,[E|_]).
member(E,[_|R]) :- member(E,R).
?- visible(+all), trace, member(X,[1,2]).

Call: (7) member(X, [1, 2])
Unify: (7) member(1, [1, 2])
Exit: (7) member(1, [1, 2]) X = 1 ;
Redo: (7) member(X, [1, 2])
Unify: (7) member(X, [1, 2])
  Call: (8) member(X, [2])
  Unify: (8) member(2, [2])
  Exit: (8) member(2, [2])
Exit: (7) member(2, [1, 2]) X = 2 ;
Backtracking in Prolog

member(E,[E|_]).
member(E,[_|R]) :- member(E,R).
?- visible(+all), trace, member(X,[1,2]).

Call: (7) member(X, [1, 2])
Unify: (7) member(1, [1, 2])
Exit: (7) member(1, [1, 2]) X = 1 ;
Redo: (7) member(X, [1, 2])
Unify: (7) member(X, [1, 2])
Call: (8) member(X, [2])
Unify: (8) member(2, [2])
Exit: (8) member(2, [2]) X = 2 ;
Exit: (7) member(2, [1, 2])
Redo: (8) member(X, [2])
Unify: (8) member(X, [2])
Call: (9) member(X, [])
Fail: (9) member(X, [])
Fail: (8) member(X, [2])
Fail: (7) member(X, [1, 2]) no.
A Meta-interpreter

```prolog
resolve(true) :- !.
```

```prolog
resolve((L,R)) :- !, resolve(L), resolve(R).
```

```prolog
resolve(T) :- clause(T, Body), write('For '), write(T), write(' need to prove '), write(Body), nl, resolve(Body).
```

```prolog
member(E,[E|_]) :- true.
```

```prolog
member(E,[_|R]) :- member(E, R).
```

```prolog
?- clause(member(X,[1,2]), B).
X = 1, B = true ; B = member(X,[2]) ; no.
```
A Meta-interpreter

\[
\text{resolve(true)} :\! :!.
\]

\[
\text{resolve((L,R)) :\! :!}, \text{resolve}(L), \text{resolve}(R).
\]

\[
\text{member}(E, [E|\_]) :\! :\text{true}.
\]

\[
\text{member}(E, [\_|R]) :\! :\text{member}(E, R).
\]

?- \text{clause}(\text{member}(X, [1,2]), B).
\]

\[
X = 1, B = \text{true} ;
\]

\[
B = \text{member}(X, [2]) ;
\]

\[
\text{no}.
\]
A Meta-interpreter

\[
\begin{align*}
\text{resolve}(\text{true}) & :- !. \\
\text{resolve}((L,R)) & :- !, \text{resolve}(L), \text{resolve}(R). \\
\text{resolve}(T) & :- \text{clause}(T, \text{Body}), \\
& \quad \text{write('For ')}, \text{write}(T), \\
& \quad \text{write(' need to prove ')}, \\
& \quad \text{write(\text{Body})}, \ \text{nl}, \\
& \quad \text{resolve}(\text{Body}).
\end{align*}
\]
A Meta-interpreter

resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).
A Meta-interpreter

```prolog
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
       write('For '), write(T),
       write(' need to prove '),
       write(Body), nl,
       resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- clause(member(X,[1,2]),B).
```
A Meta-interpreter

\[
\text{resolve}(\text{true}) \ :- \ !. \\
\text{resolve}((L,R)) \ :- \ !, \ \text{resolve}(L), \ \text{resolve}(R). \\
\text{resolve}(T) \ :- \ \text{clause}(T,\text{Body}), \\
\quad \text{write('For '), write}(T), \\
\quad \text{write(' need to prove '), write}(\text{Body}), \ \text{nl}, \\
\quad \text{resolve}(\text{Body}).
\]

\[
\text{member}(E,[E|_]) \ :- \ \text{true}. \\
\text{member}(E,[_|R]) \ :- \ \text{member}(E,R).
\]

?- \ \text{clause}(\text{member}(X,[1,2]),B). \ \ X = 1,B = \text{true} \ ; \\
\quad B = \text{member}(X,[2]) \ ; \\
\quad \text{no}.
A Meta-interpreter

resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member (1 ,[1 ,2]) need to prove true
X = 1 ;
For member (X ,[1 ,2]) need to prove member (X ,[2])
For member (2 ,[2]) need to prove true
X = 2 ;
For member (X ,[2]) need to prove member (X ,[_]).
no.
A Meta-interpreter

\[\text{resolve}(\text{true}) \leftarrow !.\]
\[\text{resolve}((\text{L}, \text{R})) \leftarrow !, \text{resolve}(\text{L}), \text{resolve}(\text{R}).\]
\[\text{resolve}(\text{T}) \leftarrow \text{clause}(\text{T}, \text{Body}),
\]
\[\quad \text{write('For '}), \text{write(} \text{T}),
\]
\[\quad \text{write(' need to prove ')),}
\]
\[\quad \text{write(} \text{Body}), \text{ nl,}
\]
\[\quad \text{resolve}(\text{Body}).\]

\[\text{member}(\text{E}, [\text{E} \mid \_]) \leftarrow \text{true}.\]
\[\text{member}(\text{E}, [\_ \mid \text{R}]) \leftarrow \text{member}(\text{E}, \text{R}).\]

?- \text{resolve(}\text{member}(\text{X}, [1, 2])).
A Meta-interpreter

```prolog
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true
```
A Meta-interpreter

resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T), write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true
X = 1 ;
A Meta-interpreter

```
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true
For member(X,[1,2]) need to prove member(X,[2])
For member(2,[2]) need to prove true
```
A Meta-interpreter

```
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
           write('For '), write(T),
           write(' need to prove '),
           write(Body), nl,
           resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true X = 1 ;
For member(X,[1,2]) need to prove member(X,[2]) X = 2 ;
For member(2,[2]) need to prove true
```
A Meta-interpreter

```prolog
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T, Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true  X = 1 ;
For member(X,[1,2]) need to prove member(X,[2])
For member(2,[2]) need to prove true  X = 2 ;
For member(X,[2]) need to prove member(X,[]).
```
A Meta-interpreter

```prolog
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).
member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true
For member(X,[1,2]) need to prove member(X,[2])
For member(2,[2]) need to prove true
For member(X,[2]) need to prove member(X,[]). no.
```
DID A PROLOG HOMEWORK QUESTION
TOOK LESS THAN 45 MINUTES PER LINE OF CODE
Outline for section 3

1 Logic in Computer Science
   - Horn clauses

2 Prolog execution model
   - Unification and backtracking
   - A Meta-interpreter

3 References
References

Max Bramer.  
*Logic Programming with Prolog.*  
ISBN 9781846282126.  

L. Sterling and E.Y. Shapiro.  
*The Art of Prolog: Advanced Programming Techniques.*  
ISBN 9780262193382.  
URL http://mitpress.mit.edu/books/art-prolog.

Patrick Blackburn, Johan Bos, and Kristina Striegnitz.  
*Learn Prolog Now!,* volume 7 of *Texts in Computing.*  
URL http://www.learnprolognow.org/.