Note on Unification in Prolog

- In Prolog, unification is used in goal matching and in the = (unify) operator.

- However, Prolog’s unification is slightly abridged: it bypasses the “occur check”: 
  \[ X = f(X) \]
  *will* unify in Prolog, but not in ordinary unification. In effect, \( X \) gets bound to the infinite term:
  \[ f(f(f(...))) \]
Checking Unifiability with Prolog

• As long as there are no occur-check violations, can use = to test.

```prolog
$ swipl
Welcome to SWI-Prolog

?- p(X, g(Z), X) = p(f(Y), Y, W).
X = f(g(Z)),
Y = g(Z),
W = f(g(Z))
```
Try These

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>mgu (or not unifiable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(X, f(X), d)$</td>
<td>$p(c, f(c), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), g(Z))$</td>
<td>$p(f(Y), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), Z)$</td>
<td>$p(g(Y), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), X)$</td>
<td>$p(f(g(h(Z))), h(Z))$</td>
<td></td>
</tr>
</tbody>
</table>
Checking Unifiability with Prover9

- In contrast to Prolog, Prover9 does use an occur-check.

<table>
<thead>
<tr>
<th>Unification succeeds</th>
<th>Unification fails due to occur-check</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-p(f(y), y).)</td>
<td>(-p(f(y), y).)</td>
</tr>
<tr>
<td>(p(x, g(z)).)</td>
<td>(p(z, g(z)).)</td>
</tr>
</tbody>
</table>

Proof:

1 \(p(x, g(y)).\)  [assumption].
2 \(-p(f(x), x).\)  [assumption].
3 \$F.  [resolve(1,a,2,a)].
Resolving Predicate Calculus Clauses

• Resolvable clauses must contain literals with the same predicate symbol but of opposite sign (one negated, the other not).

• First “rename apart” the clauses (leave no common variables).

• Pick two such literals, one from each clause.

• Determine whether the literals are unifiable, with mgu $\mu$. If they are, apply $\mu$ to all literals in both clauses. If not, the clauses don’t resolve on these particular literals.

• In the modified clauses, remove all instances of the modified literals used in unification, and form the disjunction of the remaining modified literals.
Complete Predicate Resolution Process

- The process is similar to the propositional case, except that we have to
  - **rename** apart the clauses, then
  - **unify literals** prior to resolution, and
  - **apply the mgu** to all literals in the two clauses, before obtaining the resolvent.

- There is also a special issue: “factoring”, that needs to be factored in.
Example of Predicate Resolution

- **Clauses:**
  - \( \neg \text{human}(X) \lor \text{mortal}(X) \)
  - \( \text{human}(\text{socrates}) \)
  - \( \neg \text{mortal}(\text{socrates}) \)

\[
\neg \text{human}(X) \lor \text{mortal}(X) \quad \neg \text{mortal}(\text{socrates})
\]

\[
\text{mgu } \mu = [\text{socrates}/X]
\]

\[
\neg \text{human}(\text{socrates}) \quad \text{human}(\text{socrates})
\]

\[
\text{mgu } \mu = []
\]

\[
\bot
\]
Example Resolving Predicate Clauses

- clause 1: $p(X, g(Z), X) \lor q(X, h(Z))$
- clause 2: $\neg p(f(Y), Y, W) \lor r(f(Y), g(W))$
- These are already renamed apart.

- The atoms $p(X, g(Z), X)$ vs. $p(f(Y), Y, W)$ of each unify with mgu
  
  $[f(g(Z))/X, g(Z)/Y, f(g(Z))/W]$

- Apply the mgu to both clauses:
  - clause 1': $p(f(g(Z)), g(Z), f(g(Z))) \lor q(f(g(Z)), h(Z))$
  - clause 2': $\neg p(f(g(Z)), g(Z), f(g(Z))) \lor r(f(g(Z)), g(f(g(Z))))$

- Remove the instances of the unified atoms and form the disjunction.
- Resolvent: $q(f(g(Z)), h(Z)) \lor r(f(g(Z)), g(f(g(Z))))$
Example of Predicate Resolution

- **Clauses:**
  - \( \neg p(X) \lor q(f(X), X)) \)
  - \( p(g(b)) \)
  - \( \neg q(Y, Z) \)

\[ \neg p(X) \lor q(f(X), X) \quad \neg q(Y, Z) \]

\[ [f(Z)/Y, Z/X] \]

\[ \neg p(Z) \]

\[ p(g(b)) \]

\[ [g(b)/Z] \]

\( \bot \)
Unit Preference Strategy

As with propositional resolution, resolving with unit clauses first is a good heuristic.
Check This Set for Unsatisfiability
(use Unit Preference)

1. \( \neg e(X) \lor q(X) \lor s(X, f(X)) \)
2. \( \neg e(X) \lor q(X) \lor r(f(X)) \)
3. \( p(a) \)
4. \( e(a) \)
5. \( \neg s(a, Y) \lor p(Y) \)
6. \( \neg p(X) \lor \neg q(X) \)
7. \( \neg p(X) \lor \neg r(X) \)
Clausal Form for Sequent

• Often, we’ll want to prove a sequent of the form

  \( \forall x \forall y (...) \)

  \( \forall x \forall y (...) \)

  \( \vdash \) ___

• For **premises** of the form \( \forall x \forall y (...) \) where ... has no quantifiers, we can just drop the quantifiers.

• We need to **negate** the conclusion ____.
Mushroom Example

1. Every fungus is a mushroom or a toadstool.
2. Every boletus is a fungus.
3. All toadstools are poisonous.
4. No boletus is a mushroom.
5. Specimen b is a boletus.
6. Therefore: Specimen b is poisonous.
Mushroom Example

1. $\forall X \ fungus(x) \rightarrow (mushroom(X) \lor toadstool(X))$
2. $\forall X \ boletus(X) \rightarrow fungus(X)$
3. $\forall X \ toadstool(X) \rightarrow poisonous(X)$
4. $\forall X \ boletus(X) \rightarrow \neg mushroom(X)$
5. $boletus(b)$
6. Therefore: $poisonous(b)$
Prove Unsatisfiability of Mushroom Clauses

1. \( \neg \text{fungus}(X) \lor \text{mushroom}(X) \lor \text{toadstool}(X) \)

2. \( \neg \text{boletus}(X) \lor \text{fungus}(X) \)

3. \( \neg \text{toadstool}(X) \lor \text{poisonous}(X) \)

4. \( \neg \text{boletus}(X) \lor \neg \text{mushroom}(X) \)

5. \( \text{boletus}(b) \)

6. \( \neg \text{poisonous}(b) \) \hspace{1cm} \text{(negated conclusion)}
Mushroom Clauses in Prover9

\(-fungus(x) \mid mushroom(x) \mid toadstool(x)\).
\(-boletus(x) \mid fungus(x)\).
\(-toadstool(x) \mid poisonous(x)\).
\(-boletus(x) \mid -mushroom(x)\).

\(boletus(b)\).

Goal:

\(poisonous(b)\).
Prover9 Output for Mushrooms

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PROOF %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ------- Comments from original proof -------
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 13.
% Level of proof is 5.
% Maximum clause weight is 0.
% Given clauses 0.

1 poisonous(b) # label(non_clause) # label(goal). [goal].
2 -boletus(x) | fungus(x). [assumption].
3 -fungus(x) | mushroom(x) | toadstool(x). [assumption].
4 -boletus(x) | mushroom(x) | toadstool(x). [resolve(2,b,3,a)].
5 -toadstool(x) | poisonous(x). [assumption].
6 boletus(b). [assumption].
7 -boletus(x) | -mushroom(x). [assumption].
8 -boletus(x) | mushroom(x) | poisonous(x). [resolve(4,c,5,a)].
9 mushroom(b) | poisonous(b). [resolve(8,a,6,a)].
10 -poisonous(b). [deny(1)].
11 mushroom(b). [resolve(9,b,10,a)].
12 -mushroom(b). [resolve(6,a,7,a)].
13 $F. [resolve(11,a,12,a)].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end of proof %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Sequents with $\exists$

- Often, we’ll want to prove a sequent of the form
  - $\forall x \forall y (\ldots)$,
  - $\forall x \forall y (\ldots)$
  - $\vdash \forall x \forall y (\ldots)$

- For premises of the form $\forall x \forall y (\ldots)$ where $\ldots$ has no quantifiers, we can just drop the quantifiers.

- We need to **negate** the conclusion, so that will become $\neg \forall x \forall y (\ldots)$ which is equivalent to

  $\exists x \exists y \neg (\ldots)$.

*We cannot* simply drop the $\exists$ quantifiers in this case!!
Consider the sequent

\[ \forall y \, p(y) \vdash \forall y \, p(x) \]

The premise translates to a clause

\[ p(y) \]

The conclusion is negated to become \[ \exists x \, \neg p(x) \].

How do we handle this?
Skolem Constants/Functions to the Rescue!

- To get rid of the quantifier in

\[ \exists x \, \neg p(x) \]

we use a trick:

Create a new constant, say b (called a Skolem constant) and replace x with that:

\[ \neg p(b) \]

- Some thought will show that:

There is an interpretation that satisfies \( \neg p(b) \) iff there is one that satisfies the original formula \( \exists x \, \neg p(x) \).

- We get to pick the value for b in finding a satisfying interpretation, just as we get to pick the value for x in \( \exists x \).
Resolution with Skolem Constant

- Consider the sequent
  \[ \forall y \; p(y) \vdash \forall x \; p(x) \]
- The premise translates to a clause
  \[ p(y) \]
- The negated conclusion \[ \neg \forall x \; p(x) \] translates to \[ \exists x \; \neg p(x) \].
- This gives a clause, where b is a Skolem function:
  \[ \neg p(b) \]
- We are good to go!
- Resolution produces \( \bot \) in 1 step.
Another Example

- Consider the sequent

$$\exists x \forall y \, p(x, y) \vdash \forall y \, \exists x \, p(x, y)$$

- Premise clause, with Skolem constant $b$:

$$p(b, y)$$

- Negated conclusion: $\forall y \, \exists x \, p(x, y)$ giving clause with Skolem constant $c$:

$$\neg p(x, c)$$

Resolution produces $\bot$ in 1 step.
Yet Another Example

- Consider the sequent

\[ \forall x \ a(x) \rightarrow \exists x \ b(x) \mid \neg \exists x \ (a(x) \rightarrow b(x)) \]

- Premise clause:

\[ \neg a(c) \lor b(d) \]

- Conclusion clauses (\(\neg \exists x \) becomes \(\forall x \neg \)):

\[ a(x) \]
\[ \neg b(x) \]

Resolution produces \(\bot\) in 2 steps.
Skolem Functions for the General Case

- $\forall x \forall y \ldots \exists v \ldots$

- $v$ is replaced with $f(x, y, \ldots)$

- $f$ is a new function symbol, the arguments of which are the $\forall$ quantified variables on the left.

- The rationale here is that “the $v$” that exists depends on $x, y, \ldots$.

- Again, there is an interpretation satisfying the original formula iff there is an interpretation satisfying the revised formula.
Example: Skolem with Arguments

Prove: “The composition of two onto [surjective] functions is onto.”
Example: Skolem with Arguments

- Prove: “The composition of two onto [surjective] functions is onto.”

- (Because we don’t have = yet) **represent the two functions as binary predicates.**
  F(x, y) means y is the image of x.

- “H is the composition of F and G”:

  \[ \forall x \forall y \forall z ((F(x,y) \land G(y,z)) \to H(x, z)) \land \forall x \forall z (H(x, z) \to \exists y (F(x,y) \land G(y,z))) \]

- “F is onto”: \( \forall y \exists x F(x, y) \)
- “G is onto”: \( \forall z \exists y G(y, z) \)
- “H is onto”: \( \forall z \exists x H(x, z) \)
Translation to Clausal Form

- $\forall y \exists x \ F(x, y) \text{ becomes } F(f(y), y)$ [f is a Skolem function]
- $\forall z \exists y \ G(y, z) \text{ becomes } G(g(z), z)$ [g is a Skolem function]
- $\forall x \forall y \forall z \ ((F(x,y) \land G(y,z)) \rightarrow H(x, z)) \text{ becomes }$
  $\neg F(x, y) \lor \neg G(y, z) \lor H(x, z)$
- $\forall x \forall z \ (H(x, z) \rightarrow \exists y \ (F(x,y) \land G(y,z))) \text{ becomes }$
  - $\neg H(x, z) \lor F(x, h(x, z))$ [h is a Skolem function]
  - $\neg H(x, z) \lor G(h(x, z), z)$
- $\neg \forall z \exists x \ H(x, z) \text{ becomes } \exists z \forall x \ \neg H(x, z)$, which, as a clause, is:
  - $\neg H(x, a)$ [a is a Skolem constant]
Resolution Proof

1. $F(f(x), y)$
2. $G(g(z), z)$
3. $\neg F(x, y) \lor \neg G(y, z) \lor H(x, z)$
4. $\neg H(x, z) \lor F(x, h(x, z))$
5. $\neg H(x, z) \lor G(h(x, z), z)$
6. $\neg H(x, a)$

7. $\neg F(x, y) \lor \neg G(y, a)$ from 3, 6
8. $\neg G(y, a)$ from 1, 7
9. $\bot$ from 2, 8

(3 and 4 were not needed in the proof.)
Prover9 Version

Clauses:
F(f(x), y).
G(g(z), z).
-F(x, y) | -G(y, z) | H(x, z).
-H(x, z) | F(x, h(x, z)).
-H(x, z) | G(h(x, z), z).
-H(x, a).

Proof:
1. -F(x, y) | -G(y, z) | H(x, z). [assumption].
2. F(f(x), y). [assumption].
3. -F(x, y) v -G(y, z) v H(x, z)
4. -H(x, z) v F(x, h(x, z))
5. -H(x, z) v G(h(x, z), z)
6. -H(x, a)

1. F(f(x), y)
2. G(g(z), z)
3. \neg F(x, y) \lor \neg G(y, z) \lor H(x, z)
4. \neg H(x, z) \lor F(x, h(x, z))
5. \neg H(x, z) \lor G(h(x, z), z)
6. \neg H(x, a)
How to get a clause form in general?

• First, using rules to be described, convert the formula into “prenex form” (all quantifiers are outside on the left), e.g.

\[ \forall x \ \exists y \ (F(x) \rightarrow (G(x, y) \rightarrow H(y))) \]

prefix matrix

• The parts of this form are called the “prefix” and the “matrix”.

• Skolemize \( \exists \) quantified variables.

• Drop \( \forall \) quantifiers.

• Convert the resulting matrix to CNF.
Conversion to Prenex Form

- Replace all connectives other than $\land \lor \neg$ with their $\land \lor \neg$ counterparts.

- **Push** negations inward.

- **Pull** quantifiers to the outside using the rules on the next page.
Example of Prenex Conversion

\[ \forall x \forall y (\exists z (p(x, z) \land p(y, z)) \rightarrow \exists u q(x, y, u)) \quad \text{replace} \rightarrow \]

\[ \forall x \forall y \neg (\exists z (p(x, z) \land p(y, z)) \lor \exists u q(x, y, u)) \quad \text{push} \neg \text{ in} \]

\[ \forall x \forall y ((\forall z \neg (p(x, z) \land p(y, z)) \lor \exists u q(x, y, u)) \quad \text{push} \neg \text{ in} \]

\[ \forall x \forall y (\forall z (\neg p(x, z) \lor \neg p(y, z)) \lor \exists u q(x, y, u)) \quad \text{pull} \exists u \text{ out} \]

\[ \forall x \forall y \exists u (\forall z (\neg p(x, z) \lor \neg p(y, z)) \lor q(x, y, u)) \quad \text{pull} \forall z \text{ out} \]

\[ \forall x \forall y \exists u \forall z (\neg p(x, z) \lor \neg p(y, z) \lor q(x, y, u)) \quad \text{prefix} \quad \text{matrix (already CNF in this case)} \]
Completion of Conversion to CNF

- Prenex Form:

\[ \forall x \ \forall y \ \exists u \ \forall z (\neg p(x, z) \lor \neg p(y, z) \lor q(x, y, u)) \]

- Skolemize \( u \) as \( f(x, y) \) and drop \( \forall x \ \forall y \ \forall z \)

\[ \neg p(x, z) \lor \neg p(y, z) \lor q(x, y, f(x, y)) \]
Basic Prenex Quantifier Rules
(for pulling quantifiers to the outside)

• We can show that the following replacements are equivalent.
• Here $\Rightarrow$ means “replace with”
  1. $(\forall x \ F) \land G \Rightarrow \forall x (F \land G)$, provided $x$ is not free in $G$
  2. $(\forall x \ F) \lor G \Rightarrow \forall x (F \lor G)$, provided $x$ is not free in $G$
  3. $(\exists x \ F) \land G \Rightarrow \exists x (F \land G)$, provided $x$ is not free in $G$
  4. $(\exists x \ F) \lor G \Rightarrow \exists x (F \lor G)$, provided $x$ is not free in $G$
• plus the symmetric counterparts of these rules with $G$ part quantified instead of the $F$ part.
• Renaming some variables may be need to enable the rule to be applied

• Example (with free variables in brackets):

  $(\exists x F[x]) \land \forall x G[x] \Rightarrow (by \ renaming \ second \ x)$
  $(\exists x F[x]) \land \forall y G[y] \Rightarrow (by \ rule \ 3, \ as \ x \ is \ not \ free \ in \ G)$
  $\exists x (F[x] \land \forall y G[y]) \Rightarrow (by \ rule \ 1 \ symmetric \ counterpart)$
  $\exists x \forall y (F[x] \land G[y])$
Justifying Prenex Quantifier Rules Using Natural Deduction: $\forall \land$ Rule a.

Proviso is introduced by prefixing ‘WHERE x NOTIN G IS’ in Jape.
To establish **equivalence**, rather than just implication, we need the **converse** of each rule: $\forall \land$ Rule b.

Need the **non-empty universe** assumption in this direction (not implicit in Jape).

Otherwise, there is no way to get G by itself.
Justification of Rules
Using Natural Deduction: $\forall \lor \; \text{Rule a.}$
Justification of Rules
Using Natural Deduction: $\forall \lor$ Rule b.
Justification of Rules Using Natural Deduction: \( \exists v \ a. \)

Non-empty universe assumption, needed in 7-11
Justification of Rules
Using Natural Deduction: \( \exists \forall b. \)
Justification of Rules
Using Natural Deduction: $\exists x \wedge a$. 

```
1: $\exists x. F(x) \wedge G$  premise
2: $G$  \wedge elim 1
3: $\exists x. F(x)$  \wedge elim 1
4: actual i, $F(i)$  assumptions
5: $F(i) \wedge G$  \wedge intro 4.2, 2
6: $\exists x. (F(x) \wedge G)$  $\exists$ intro 5, 4.1
7: $\exists x. (F(x) \wedge G)$  $\exists$ elim 3, 4–6

Provided:
$\exists$ NOTIN G
```
Justification of Rules
Using Natural Deduction: $\exists x \wedge b$. 

\[
\begin{align*}
1: & \exists x. (F(x) \wedge G) \quad \text{premise} \\
2: & \text{actual } i_1, F(i_1) \wedge G \quad \text{assumptions} \\
3: & F(i_1) \quad \wedge \text{elim } 2.2 \\
4: & \exists x. F(x) \quad \exists \text{intro } 3,2.1 \\
5: & \exists x. F(x) \quad \exists \text{elim } 1,2-4 \\
6: & \text{actual } i, F(i) \wedge G \quad \text{assumptions} \\
7: & G \quad \wedge \text{elim } 6.2 \\
8: & G \quad \exists \text{elim } 1,6-7 \\
9: & \exists x. F(x) \wedge G \quad \wedge \text{intro } 5,8
\end{align*}
\]
Example of Conversion by Prover9

Input for the “onto” example:
all y exists x F(x, y).
all z exists y G(y, z).
all x all y all z (F(x, y) & G(y, z) -> H(x, z)).
all x all z (H(x, z) -> exists y (F(x, y) & G(y, z))).
goal: all z exists x H(x, z).

Proof:
1 (all x exists y F(y,x)) # label(non_clause). [assumption].
2 (all x exists y G(y,x)) # label(non_clause). [assumption].
3 (all x all y all z (F(x,y) & G(y,z) -> H(x,z))) # label(non_clause). [assumption].
5 (all x exists y H(y,x)) # label(non_clause) [goal].
6 -F(x,y) | -G(y,z) | H(x,z). [clausify(3)].
7 F(f1(x),x). [clausify(1)].
9 -G(x,y) | H(f1(x),y). [resolve(6,a,7,a)].
10 G(f2(x),x). [clausify(2)].
13 -H(x,c1). [deny(5)].
14 H(f1(f2(x)),x). [resolve(9,a,10,a)].
15 $F$. [resolve(14,a,13,a)].
Example: Group Theory Clauses

- f is the group operation, i is the inverse operation, e is the equality predicate

- $\forall x \forall y \forall x \ e(f(x, f(y, z)), f(f(x, y), z))$
  becomes
  $e(f(x, f(y, z)), f(f(x, y), z))$

- $\forall x \ e(f(x, u), x)$
  becomes
  $e(f(x, u), x)$

- $\forall x \ e(f(x, i(x)), u)$
  becomes
  $e(f(x, i(x)), u)$
Example: Equality Theory Clauses

- We need to axiomatize equality predicate e, e.g.

- $\forall x \ e(x, x)$ becomes
  $e(x, x)$

- $\forall x \ \forall y \ \forall u \ \forall v \ ((e(x, y) \land e(v, w)) \rightarrow e(f(x, v), f(y, w)))$ becomes
  $\neg e(x, y) \lor \neg e(v, w) \lor e(f(x, v), f(y, w))$

- $\forall x \ \forall u \ (e(x, u) \rightarrow e(i(x), i(u)))$ becomes
  $\neg e(x, u) \lor e(i(x), i(u))$

- $\forall x \ \forall y \ (e(x, y) \rightarrow e(y, x))$ becomes
  $\neg e(x, y) \lor e(y, x)$

Exercise: Convert the transitive property of e.
Example of Group Theory Clauses with Negated Conclusion

1. \( e(f(x, f(y, z)), f(f(x, y), z)) \)
2. \( e(f(x, u), x) \)
3. \( e(f(x, i(x)), u) \)
4. \( e(x, x) \)
5. \( \neg e(x, y) \lor \neg e(v, w) \lor e(f(x, v), f(y, w)) \)
6. \( \neg e(x, y) \lor e(y, x) \)
7. \( \neg e(x, y) \lor \neg e(y, z) \lor e(x, z) \)
8. \( \neg e(i(i(b)), b) \)

This is to show that \( \forall x \ e(i(i(x)), x) \):

“In a group, the inverse of the inverse of an element is the element itself.”
Prover9 input using builtin equality

\[ f(x, f(y, z)) = f(f(x, y), z). \]
\[ f(x, c) = x. \]
\[ f(x, i(x)) = c. \]
\[ i(i(b)) \neq b. \]

(Show unsatisfiable)
Prover9 proof using builtin equality

1. \( f(x,f(y,z)) = f(f(x,y),z) \). [assumption].
2. \( f(f(x,y),z) = f(x,f(y,z)) \). [copy(1),flip(a)].
3. \( f(x,c) = x \). [assumption].
4. \( f(x,i(x)) = c \). [assumption].
5. \( i(i(b)) \neq b \). [assumption].
6. \( f(x,f(c,y)) = f(x,y) \). [para(3(a,1),2(a,1,1),flip(a)].
7. \( f(x,f(i(x),y)) = f(c,y) \). [para(4(a,1),2(a,1,1),flip(a)].
12. \( f(c,i(i(x))) = x \). [para(4(a,1),7(a,1,2),rewrite([3(2)]),flip(a)].
14. \( f(x,i(i(y))) = f(x,y) \). [para(12(a,1),2(a,2,2),rewrite([3(2)])].
15. \( f(c,x) = x \). [para(12(a,1),6(a,2),rewrite([14(5),6(4)])].
18. \( f(x,f(i(x),y)) = y \). [back_rewrite(7),rewrite([15(5)])].
21. \( i(i(x)) = x \). [para(4(a,1),18(a,1,2),rewrite([3(2)]),flip(a)].
22. \( F \). [resolve(21,a,5,a)].
Equality: Paramodulation

- Prover9 has a built-in equality, so axiomatizing equality as a predicate is not generally necessary.

- The “paramodulation” rule, which essentially captures the $=$ rules of Natural Deduction.

$$
\frac{\alpha \lor (s = t) \quad \beta \lor \gamma[r]}{(\alpha \lor \beta \lor \gamma[t])\theta}
\quad \gamma[r] \text{ is a literal containing term } r
\quad \theta = \text{unify}(s,r)
$$
Example: Non-Clausal Input in Prover9: Automatic Translation to Clausal Form

all x exists y r(x, y).

all x all y all z ((r(x,y)&r(y,z))->r(x,z)).

all x all y (r(x,y) -> r(y,x)).

-(all x r(x,x)).
Example: Non-Clausal Input in Prover9: Automatic Translation to Clausal Form

becomes (c1, f1 are Skolem constant and function):

1 (all x exists y r(x,y)) # label(non_clause).
2 (all x all y all z (r(x,y) & r(y,z) -> r(x,z))) # label(non_clause).
3 (all x all y (r(x,y) -> r(y,x))) # label(non_clause).
4 -(all x r(x,x)) # label(non_clause).

5 r(x,f1(x)).
6 -r(x,y) | -r(y,z) | r(x,z).
7 -r(x,y) | r(y,x).
8 -r(c1,c1).

10 r(f1(x),x).
12 -r(f1(c1),c1).
13 $F.$
Answer Extraction

- Resolution is not just for proving theorems anymore.

- Resolution can be used for extracting answers from a database, knowledge base, or reasoning system.
From Yes-No Answer to Terms

- Consider the clause set:
  - $\neg \text{human}(x) \lor \text{mortal}(x)$
  - $\text{human}(\text{socrates})$
  - $\neg \text{mortal}(\text{socrates})$

- Obviously this set is unsatisfiable, and a proof can be obtained by resolution.

- What if we drop the third clause. The first two clauses are satisfiable, and can be considered a “knowledge base”.

- We can ask a question of this knowledge base, such as:
  
  Name someone who is mortal.
Answer Literals

• An answer literal is a special literal that captures the answer to a question.

• We convert the negation of a specific conclusion into a clause involving an answer literal:

\[
\neg\text{mortal}(x) \lor \text{answer}(x).
\]

• Resolution stops when a clause with only the answer is present.
Resolution with Answer Literals

- **Clauses:**
  1. \( \neg \text{human}(x) \lor \text{mortal}(x) \)
  2. \( \text{human}(\text{socrates}) \)
  3. \( \neg \text{mortal}(x) \lor \text{answer}(x) \)

- **Resolution steps:**
  4. \( \text{mortal}(\text{socrates}) \) from 1, 2
  5. \( \text{answer}(\text{socrates}) \) from 3, 4
Answering Questions in Prover9

- To find terms such that $p(x)$, incant:

  $$-p(x) \# \text{answer}(x).$$
Who is Mortal, in Prover9

Clausal Form

- \text{human}(x) \mid \text{mortal}(x).

\text{human}(\text{socrates}).

\text{mortal}(x) \not= \text{answer}(x).
Prover9 Solution with Answer

Clauses

- $\text{human}(x) \lor \text{mortal}(x)$.

- $\text{human}(\text{socrates})$.

- $\text{-mortal}(x) \# \text{answer}(x)$.

Proof
Who is Caroline’s Grandfather?

**Clauses**

- father(x, y) | parent(x, y).
- father(x, y) | -parent(y, z) | grandfather(x, z).

father(joe, john).
father(john, caroline).

-grandfather(x, caroline) # answer(x).

**Proof**

1. father(joe, john). [assumption].
2. -father(x, y) | parent(x, y). [assumption].
3. -father(x, y) | -parent(y, z) | grandfather(x, z). [assumption].
4. father(john, caroline). [assumption].
5. -parent(john, x) | grandfather(joe, x). [resolve(1, a, 3, a)].
6. -grandfather(x, caroline) # answer(x). [assumption].
7. -parent(john, caroline) # answer(joe). [resolve(5, b, 6, a)].
8. parent(john, caroline). [resolve(4, a, 2, a)].
9. $F$ # answer(joe). [resolve(8, a, 10, a)].
Logic Puzzles solvable by Resolution

% Professors Dodds, Lewis, and Stone each frequent different establishments (one of Alice's, Harry's, or Joe's) for liquid refreshment.

% Each prof prefers a different beer (one of Anchor, Bud, and Miller)
% Each establishment serves a unique beer.

% Professor Stone prefers Bud.
% Professor Lewis doesn't prefer Miller.
% The prof who prefers Miller frequents Alice's bar.
% The prof who prefers Anchor does not frequent Joe's.

% Which bar does each prof frequent and what beer does each prefer?
Encoding Information & Query

% Clauses corresponding to the clues:

prefer(Stone, Bud).                                      % Clue 1
-prefer(Lewis, Miller).                                 % Clue 2
-prefer(x, Miller) | frequent(x, Alice).          % Clue 3
-prefer(x, Anchor) | -frequent(x, Joe).          % Clue 4

% Individuals

prof(Dodds). prof(Stone). prof(Lewis).
beer(Anchor). beer(Bud). beer(Miller).
bar(Alice). bar(Harry). bar(Joe).

% Although constants do not unify, they could conceivably be equal.
Dodds != Stone. Dodds != Lewis. Stone != Lewis.

% Any bar that a professor frequents serves the beers that he or she prefers.
-frequent(x, y) | serves(y, z) | prefer(x, z).

% Every bar is frequented by some prof.
-bar(y) | frequent(Dodds, y) | frequent(Stone, y) | frequent(Lewis, y).

% Every beer is preferred by some prof.
-beer(y) | prefer(Dodds, y) | prefer(Stone, y) | prefer(Lewis, y).

% Each bar serves a unique beer.
-serves(x, y) | -serves(x, z) | y = z.

% Each prof prefers a unique beer.
-prefer(x, y) | -prefer(x, z) | y = z.

% Each prof frequents a unique bar.
-frequent(x, y) | -frequent(x, z) | y = z.

% Which bars are frequented, and which beers preferred, by which professors?

-frequent(Dodds, x) | -frequent(Stone, y) | -frequent(Lewis, z)
| -prefer(Dodds, u) | - prefer(Stone, v) | -prefer(Lewis, w)
#answer([Dodds, x, u, Stone, y, v, Lewis, z, w]).
Prover9 Solution

Dodds-Alice-Miller
Stone-Joe-Bud
Lewis-Harry-Anchor
What if Solution not Unique?

- I’m not quite sure how Prover9 handles this, if it indeed can.

- It’s predecessor, Otter, could handle it by showing the alternatives as a disjunction of answer literals.
State and Motion Puzzles and Games

- Moves in a motion puzzle or game can often be encoded as logic.

- Resolution can be used to find a solving or winning sequence of moves.
Example: Linear Peg Solitaire
Linear Peg Solitaire Objective

- Pegs of two colors are shown in their home positions at the top.
- The objective is to completely reverse the pegs, so that each peg’s original home is occupied by a peg of the opposite color.
- Allowable actions:
  - Move: A peg can be moved toward the opposite side by moving into an adjacent empty hole.
  - Jump: A peg can jump toward the opposite side over a peg of the opposite color, provided that there is a hole to receive the jumping peg.
- Versions of the puzzle exists for $2n$ pegs (n of each color) and $2n+1$ holes.
- Ideally, each version can be solved.
Formulation (This will be important when we talk about computability later on.)

- Represent the state of the game with two terms.
- Say the pegs are w for white, r for red.
- Represents the pegs away from the hole in either direction as a composition of function symbols.
  - The initial state shown is: 
    \[ s(w(w(w(w(c))))), r(r(r(r(c)))) \]
  - The second state shown is: 
    \[ s(r(w(r(w(w(c)))))), w(r(r(c))) \]
  - c is a dummy constant symbol
Formulating Moves

• **Simple moves (non-jump):**
  - move(s(w(X), Y), s(X, w(Y)))) (wm)
  - move(s(X, r(Y)), s(r(X), Y)) (rm)

• **Jump moves:**
  - move(s(r(w(X)), Y), s(X, r(w(Y)))) (wj)
  - move(s(X), w(r(Y))), s(w(r(X)), Y)) (rj)
Formulating Reachability

- Initial state:
  \( \text{reachable}(s(w(w(w(w(c))))), r(r(r(r(c)))))) \)

- State change:
  \( \neg \text{reachable}(X) \lor \neg \text{move}(X, Y) \lor \text{reachable}(Y) \)

- Final state:
  \( \neg \text{reachable}(s(r(r(r(r(c))))), w(w(w(w(c))))) \).
Prover9 Version

move(s(w(x), y), s(x, w(y))).
move(s(x, r(y)), s(r(x), y)).
move(s(r(w(x)), y), s(x, r(w(y))).
move(s(x, w(r(y))), s(w(r(x)), y)).

reachable(s(w(w(w(w(c)))), r(r(r(r(c)))))).

-reachable(x) | -move(x, y) | reachable(y).

-reachable(s(r(r(r(r(c)))))), w(w(w(w(c)))))

Proof for 2 pegs of each color

1 -reachable(x) | -move(x,y) | reachable(y).
2 move(s(r(x),y),s(x,r(y)))).
3 move(s(x,b(y)),s(b(x),y)).
5 move(s(b(r(x)),y),s(x,b(r(y))))).
6 move(s(x,r(b(y))),s(r(b(x)),y)).
8 reachable(s(r(r(c)),b(b(c))))).
9 -reachable(s(b(b(c)),r(r(c))))).
10 -reachable(s(r(x),y)) | reachable(s(x,r(y))).
11 -reachable(s(x,b(y))) | reachable(s(b(x),y)).
13 -reachable(s(b(r(x)),y)) | reachable(s(x,b(r(y))))).
14 -reachable(s(x,r(b(y)))) | reachable(s(r(b(x)),y)).
16 -reachable(s(r(b(b(c))),r(c))).
17 reachable(s(r(c),r(b(b(c))))).
22 -reachable(s(b(c),r(b(r(c)))))
25 -reachable(s(c,b(r(b(r(c))))))
28 reachable(s(r(b(r(c))),b(c))).
31 reachable(s(b(r(b(r(c)))),c)).
33 -reachable(s(b(r(c)),b(r(c))))
37 -reachable(s(b(r(b(r(c)))),c)).
38 $F. [resolve(37,a,31,a)].
Proof for 3 Pegs of Each Color
(output of Otter rather than Prover9, more traceable)

7 [ ]  \text{reachable}(s(w(w(w(c)))), r(r(r(c))))).
8 [hyper, 7, 1, 4]  \text{reachable}(s(r(w(w(w(c)))), r(r(c))))).
12 [hyper, 5, 1, 8]  \text{reachable}(s(w(w(c)), r(w(r(r(r(c))))))).
16 [hyper, 12, 1, 3]  \text{reachable}(s(w(c), w(r(w(r(r(c))))))).
20 [hyper, 16, 1, 6]  \text{reachable}(s(w(r(w(c))), w(r(r(c))))).
27 [hyper, 20, 1, 6]  \text{reachable}(s(w(r(w(r(w(c))))), r(c))).
34 [hyper, 27, 1, 4]  \text{reachable}(s(r(w(r(w(r(w(c)))))), c)).
39 [hyper, 34, 1, 5]  \text{reachable}(s(r(w(r(w(c)))), r(w(c))))).
44 [hyper, 39, 1, 5]  \text{reachable}(s(r(w(c)), r(w(r(w(c)))))).
51 [hyper, 44, 1, 5]  \text{reachable}(s(c, r(w(r(w(r(w(c))))))).
57 [hyper, 51, 1, 4]  \text{reachable}(s(r(c), w(r(w(r(w(c))))))).
63 [hyper, 57, 1, 6]  \text{reachable}(s(w(r(r(c))), w(r(w(c))))).
69 [hyper, 63, 1, 6]  \text{reachable}(s(w(r(w(r(r(c))))), w(c))).
72 [hyper, 69, 1, 3]  \text{reachable}(s(r(w(r(r(c)))), w(w(c))))).
75 [hyper, 72, 1, 5]  \text{reachable}(s(r(r(c)), r(w(w(w(c)))))).
77 [hyper, 75, 1, 4]  \text{reachable}(s(r(r(r(c))), w(w(w(c))))).
78 [binary, 77.1, 2.1] \text{\$F\$}.
Pegs vs. # of Moves in Solution

<table>
<thead>
<tr>
<th>Pegs of Each Color</th>
<th># of Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>n</td>
<td>(n^2 + 2n)</td>
</tr>
</tbody>
</table>
Determining the Move Sequence

- The previous proofs only showed that the puzzle could be solved for those variations.

- The actual move sequence would have to be dug out from the proof steps.

- We can modify the rules so that the move sequence is obtained as a byproduct.
Determining the Move Sequence

- Use function composition to represent accumulated move sequence.
- Revised rules (4-pegs, where specific):
  - move(s(w(x), y), s(x, w(y)), z, wm(z)).
  - move(s(x, r(y)), s(r(x), y), z, rm(z)).
  - move(s(r(w(x)), y), s(x, r(w(y))), z, wj(z)).
  - move(s(x, w(r(y))), s(w(r(x)), y), z, rj(z)).
  - reachable(s(w(w(w(w(c)))))), r(r(r(r(c))))), d).
  - -reachable(x, z) | -move(x, y, z, zz) | reachable(y, zz).
  - -reachable(s(r(r(r(r(c))))), w(w(w(w(c))))), z) #answer(z).
Move Sequence Read Inside-Out

- For 4 pegs of each color:
  \[
  \text{answer}(\text{rm}(\text{wj}(\text{wm}(\text{rj}(\text{rj}(\text{rm}(\text{wj}(\text{wj}(\text{wm}(\text{rj}(\text{rj}(\text{rj}(\text{wm}(\text{wj}
  (\text{wj}(\text{wj}(\text{rm}(\text{wj}(\text{rm}(\text{wm}(\text{wj}(\text{rm}(d)))))))))))))))))))))))).
  \]

- The sequence is:
  \[
  \begin{align*}
  &\text{rm }\text{wj }\text{wm }\text{rj }\text{rj }\text{rm }\text{wj }\text{wj }\text{wj }\text{wm }\text{rj }\text{rj} \\
  &\text{rj }\text{rj }\text{wm }\text{wj }\text{wj }\text{wj }\text{rm }\text{rj }\text{rj }\text{wm }\text{wj }\text{rm}
  \end{align*}
  \]

- (For this puzzle, the move sequence is coincidentally a palindrome.)
Herbrand’s Theorem: Why Resolution Works

• Jacques Herbrand showed the following (1930):
  • A set of clauses is unsatisfiable iff there is a refutation using a certain kind of **symbolic interpretation** (known as a **Herbrand Interpretation**):
    • For each constant symbol, the interpretation is literally that symbol. (If no constant symbols, add one.)
    • Functions are defined as follows:
      \[ I(f(t_1, ..., t_n)) = \text{the string } f(I(t_1), ..., I(t_n)) \]

http://mathworld.wolfram.com/HerbrandUniverse.html
http://en.wikipedia.org/wiki/Herbrand_interpretation
Ground Terms and Clauses

- A **ground term** is a term in which there are no variables.

- A **ground clause** is a clause in which there are no variables.

- **Herbrand** showed: If there is a refutation at all, there is one using only ground clauses from a Herbrand interpretation.
Other Fine Points of Resolution

- Treat clauses as sets (reduce when necessary).
- Factoring may be necessary.
Remember to treat clauses as sets.

- \( q(b, X) \lor p(X) \lor q(b, a) \)
- \( \neg q(Y, a) \lor p(Y) \)
- These are already renamed apart.

- unify \( q(b, X) \) with \( \neg q(Y, a) \)
- mgu is \([a/X, b/Y]\]

- Modified clauses:
  - \( q(b, a) \lor p(a) \lor q(b, a) \)
  - \( \neg q(b, a) \lor p(b) \)

- There are two instances of \( q(b, a) \) in the first clause; both are removed in resolving.
  - Resolvent: \( p(a) \lor p(b) \)
Binary Resolution and Factoring

- What we have seen so far is “binary” resolution — unifying two literals to achieve a resolvent.

- In general, binary resolution is not enough.

- We might need to “factor” two or more literals in the same clause to make progress.
Factoring

- Two or more literals of the same sign in one clause can be **unified** (before renaming apart) so that the resulting literals can be collapsed into one.

- The resulting clause is called a **factor** of the original.

- The factor (with all variables quantified) is logically **implied by** the more-general original (with all variables quantified).
Factoring Example

• Consider the clause:

\[ P(x) \lor P(f(y)) \lor \neg Q(x) \]

• The first two literals can be unified using the substitution \([f(y)/x]\).

• The resulting factor is:

\[ P(f(y)) \lor \neg Q(f(y)) \]

• \((\forall x \, \forall y \, (P(x) \lor P(f(y)) \lor \neg Q(x))) \rightarrow \forall y \, (P(f(y)) \lor \neg Q(f(y)))\) is valid.
Use of Factoring

• Suppose our clause set includes:

\[ P(x) \lor P(f(y)) \lor \neg Q(x) \]
\[ \neg P(f(a)) \]

• With binary resolution, we’d get the resolvent:

\[ P(x) \lor \neg Q(x). \]

• If we **first factor**, to get \( P(f(y)) \lor \neg Q(f(y)) \) as on the previous slide, we can get a resolvent \( \neg Q(f(a)) \), which is more helpful (as a unit clause).
Full Resolution of Two Clauses

- Binary resolution of the clauses.
- Binary resolution of one clause with a factor of the other.
- Binary resolution of factors of both clauses.
Case Where Factoring is Necessary

- $P(x) \lor P(y)$
- $\neg P(a) \lor \neg P(b)$
- Without factoring, generate:
  - $P(y) \lor \neg P(b)$
  - $P(x) \lor \neg P(a)$
- and more similar clauses, but never the empty clause.
Case Where Factoring is Necessary

- $P(x) \lor P(y)$
- $\neg P(a) \lor \neg P(b)$
- With factoring, get factor $p(x)$ from first clause, then generate:
  - $\neg P(b)$
  - $\bot$
Subsumption

- A clause $C$ **subsumes** a clause $D$ if there is a substitution $\theta$ such that $C\theta \subseteq D$, where we interpret the clauses as **sets** of their literals.

- If a clause $D$ in a set of clauses is subsumed by another clause $C$ **within the set**, then we can delete $D$ from the set without affecting the case of whether the empty clause $\bot$ is derivable.

The subsuming clause is more general.
Subsumption Examples

• $P(X)$ subsumes $P(X) \lor Q(Y)$ by the empty substitution $[]$.

• $\neg P(X) \lor Q(f(X))$ subsumes
  
  $\neg P(Z) \lor \neg P(h(Y)) \lor Q(f(h(Y)))$
  
  by the substitution $[h(Y)/X, h(Y)/Z]$. 