Housekeeping: Reduced Clause Sets

- A clause set is **reduced** provided:
  - No literal occurs multiple times in any clause.
    - $p \lor \neg q \lor p$ is disallowed in a reduced set.
  - No clause contains a literal and its negation.
    - $p \lor q \lor \neg p$ is disallowed in a reduced set.

- Any clause set $S$ is equivalent to a reduced set $\text{reduce}(S)$:
  - Replace multiple occurrences of a literal with a **single occurrence** of the literal.
  - **Drop** any clauses containing a literal and its negation.
    (Such clauses are equivalent to $T$, and do thus do not affect satisfiability of the set of clauses.)
  - Replace multiple occurrences of a clause (as a set) with a single occurrence.
reduce example

\[
\text{reduce(}\{p \lor \neg q \lor p, \\
p \lor q \lor \neg p \lor q\}\text{)} = \\
\{p \lor \neg q\}
\]
Resolution Method

- **Input**: A reduced set of clauses.

- **Output**: A set of clauses equivalent to the input set, such that the original set is unsatisfiable iff the final set contains the empty clause $\bot$.

- There is no valuation that satisfies $\bot$ (much less $\bot$ together with other clauses).
How Resolution Works

- Do Repeatedly, until no further steps can be taken:
  - From the set of clauses, pick a pair from which a new clause, called the “resolvent”, can be created. (Must resolve the pair to find this out.)

  - Add the resolvent to the set.

  - If \( \bot \) is ever added to the set, stop. The original set of clauses is unsatisfiable.

- Conversely, if the original set of clauses is unsatisfiable, it is possible to eventually derive \( \bot \).
What is the Resolvent?

- Suppose \( p \) is a proposition symbol.

- If the set contains clauses of both forms
  - \( p \lor \varphi \)
  - \( \neg p \lor \psi \)

- where \( \varphi \) and \( \psi \) are clauses (either could be empty), then the resolvent is the reduced version of
  \[ \varphi \lor \psi. \]

- \( p \) and \( \neg p \) are said to be “clashing” literals.
Resolution as a Deduction Rule

\[ \frac{p \lor \varphi}{\neg p \lor \psi} \]

\[ \varphi \lor \psi \] (in reduced form)

where \( p \) is any proposition symbol and \( \varphi \) and \( \psi \) are clauses (possibly empty).
Example of Resolvents

- Consider the clauses
  - $p \lor \neg q \lor \neg s$
  - $q \lor r \lor \neg s$

- A resolvent (based on literals $q$, $\neg q$) is:
  - $p \lor r \lor \neg s$
Example of Resolvents

• Consider the clauses
  • \( p \lor r \)
  • \( \neg r \)

• The resolvent is:
  • \( p \)
Example of Resolvents

• Consider the clauses
  • $p$
  • $\neg p$

• Since $p$ and $\neg p$ occur in different clauses, the resolvent is:
  • $\bot$
Example of Resolvents

- Consider the clauses
  - $p \lor \neg q \lor r$
  - $q \lor \neg r \lor \neg s$
- One resolvent (based on literals $q, \neg q$) is:
  - $p \lor r \lor \neg r \lor \neg s$
- Another (based on literals $r, \neg r$) is:
  - $p \lor q \lor \neg q \lor \neg s$
- Both of these would be **dropped** in reducing, however, since each contains a literal and its negation.
Resolution Algorithm

• Start with a set $S$ of reduced clauses.

• while $S$ does not contain $\bot$ and the following step adds something new to $S$:
  
  • Add to $S$ the resolvent $R$ of any two clauses such that $R$ is not already in $S$ and the resolvent does not contain complementary literals.

• The original $S$ is unsatisfiable iff $\bot$ is in $S$. 
Unit Clauses

- A clause with exactly one literal is called a **unit clause**.

- The ultimate step in resolving to $\bot$ will be to resolve two unit clauses.

- Resolving a unit clause with a clause having $n > 0$ literals results in a clause with fewer than $n$ literals.
Unit Preference Strategy

- Preferring unit clauses is a good heuristic.
Example 1 (Highlighting unit clauses)

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r\}$
  resolve $p \lor \neg q$ with $\neg p$, adding $\neg q$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r, \ \neg q\}$
  resolve $q \lor r$ with $\neg q$, adding $r$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r, \ \neg q, \ r\}$
  resolve $\neg r$ with $r$ adding $\bot$ to $S$.

- Stop $\bot \in S$.

- The original $S$ is unsatisfiable, as $\bot \in S$. 

Example 2

- \( S = \{p \lor \neg q, \ q \lor r, \ \neg p\} \)
  Resolve \( p \lor \neg q \) with \( \neg p \), adding \( \neg q \) to \( S \).

- \( S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q\} \)
  Resolve \( q \lor r \) with \( \neg q \), adding \( r \) to \( S \).

- \( S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q, \ r\} \)
  Resolve \( p \lor \neg q \) with \( q \lor r \), adding \( p \lor r \) to \( S \).

- \( S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q, \ r, \ p \lor r\} \)
  Stop. No new resolvents are possible. The original set is satisfiable, as \( \bot \notin S \).
**Soundness:** Any valuation satisfying both \( p \lor \varphi \) and \( \neg p \lor \psi \) satisfies \( \varphi \lor \psi \).

- Suppose \( \nu \) satisfies both \( p \lor \varphi \) and \( \neg p \lor \psi \).

- Either \( \nu(p) = T \) or \( \nu(p) = F \).

- If \( \nu(p) = T \), then \( \nu(\neg p) = F \). In order to satisfy \( \neg p \lor \psi \) then, \( \nu(\psi) = T \). Thus \( \nu(\varphi \lor \psi) = T \).

- If \( \nu(p) = F \), in order to satisfy \( p \lor \varphi \), \( \nu(\varphi) = T \). Thus \( \nu(\varphi \lor \psi) = T \).

- Thus adding the resolvent preserves the valuations that satisfy the set of clauses.
Completeness

- Completeness is more complicated and we will not prove it here.

- We’d have to show that if a set is unsatisfiable, there is a set of resolution steps that result in the empty clause.

Resolution Algorithm Termination (propositional case)

- Closure is always achievable.

- The set of distinct reduced clause sets for a given set of proposition symbols is finite.

- At worst, every possible clause (regarding reordering of symbols as equivalent) will be generated.

- How many distinct clauses can there be for n proposition symbols?
Resolution in tabular form

1. $p \lor \neg q$  Premise
2. $q \lor r$  Premise
3. $\neg r$  Premise
4. $\neg p$  Premise
5. $q$  Resolution 2, 3
6. $p$  Resolution 1, 5
7. $\bot$  Resolution 6, 4
Resolution as a Tree

\[ p \lor \neg q \quad q \lor r \quad \neg r \quad \neg p \]

children nodes are resolvents
Try resolving these clause sets:

- \neg p \lor \neg q \lor \neg r,
  \neg q \lor r,
  q \lor s,
  \neg s,
  p

- p \lor \neg q \lor r,
  q \lor r,
  \neg p
Sometimes a DAG is more appropriate than a tree for showing all options.

We avoid identifying the two \( \perp \) nodes, so as not to confuse the two sets of antecedents.
Useful Resolution Short-cuts

- **Uncomplemented Literal Lemma** (also called the “Purity Rule”)

  If a **literal** appears in one or more clauses, but its **complement appears in no clause**, then every clause containing that literal can be deleted from the set without changing satisfiability.

- **Rationale**: The literal in question can be assigned T without loss of generality, thus clauses containing it cannot affect satisfiability.
Example of Uncomplemented Literal Lemma

- \( \neg p \lor q \lor r, \)
  - \( \neg q \lor r, \)
  - \( q \lor s, \)
  - \( \neg s, \)
  - \( p \)
- \( r \) occurs only uncomplemented.
- The clause set is unsatisfiable iff the following set is:
  - \( q \lor s, \)
  - \( \neg s, \)
  - \( p \)
- and this set is unsatisfiable iff \( \neg s \) is unsatisfiable (which it isn’t).
Further Resolution Short-Cuts

- **Unit Clause Lemma:**

  If a **unit** cause (clause with only one literal L) exists within the set, the following operation may be performed without affecting satisfiability:

  - Remove all clauses containing L.
  - Remove the complement of L from all remaining clauses.

- **Rationale:** The literal in question **must** be assigned T in a satisfying interpretation. Hence all clauses containing it will be T and contribute nothing to the set. Likewise, its complement must be assigned F, and thus contribute nothing to the individual clauses.
Example of Unit Clause Lemma

- \( \neg p \lor q \lor r \)
  - \( q \lor s \)
  - \( \neg s \)
  - \( p \lor \neg s \)

- \( \neg s \) is a unit clause. The complement of \( \neg s \) is \( s \).

- The clause set is unsatisfiable iff the following set is:
  - \( \neg p \lor q \lor r \)
  - \( q \)
  - \( (\text{formerly } q \lor s) \)

(\( \neg s \) and \( p \lor \neg s \) were removed.)
The previous two edit rules are the basis of another algorithm for satisfiability: DPLL for Davis-Putnam-Logemann-Loveland

Further Useful Optimizations

Subsumption Lemma:

- One clause **subsumes** another if the former’s literals are a **subset** of the latter’s.

- If one clause of a set subsumes another, the **subsumed** clause (the larger one) can be **dropped** from the set.

- **Rationale**: If C subsumes D, then any interpretation satisfying C must also satisfy D (because the literals are disjoined). Thus the satisfiability of the set of clauses is unaffected if D is removed.
Example of Subsumption Lemma

- \( \neg p \lor q \lor \neg r, \)
  \( \neg p \lor \neg r, \)
  \( p \lor r \lor q \)
- The second clause subsumes the first.
- The clause set is unsatisfiable iff the following set is:
  \( \neg p \lor \neg r, \)
  \( p \lor r \lor q \)
Common Special Case of Clause Set

• Often we want to prove a sequent such as:
  \[
  \varphi_{11} \land \varphi_{12} \land \ldots \land \varphi_{m1} \rightarrow \psi_1,
  \varphi_{21} \land \varphi_{22} \land \ldots \land \varphi_{m2} \rightarrow \psi_2,
  \ldots
  \varphi_{n1} \land \varphi_{n2} \land \ldots \land \varphi_{mn} \rightarrow \psi_n
  \models \chi_1 \land \chi_2 \land \ldots \land \chi_p
  \]
  where each symbol represents a literal.

• This can be done by showing that the following clause set is unsatisfiable:
  \[
  \{ \neg \varphi_{11} \lor \neg \varphi_{12} \lor \ldots \lor \neg \varphi_{m1} \lor \psi_1,
  \neg \varphi_{21} \lor \neg \varphi_{22} \lor \ldots \lor \neg \varphi_{m2} \lor \psi_2,
  \ldots
  \neg \varphi_{n1} \lor \neg \varphi_{n2} \lor \ldots \lor \neg \varphi_{mn} \lor \psi_n,
  \neg \chi_1 \lor \neg \chi_2 \lor \ldots \lor \neg \chi_p \}
  \]
Strategic Optimizations

- **Unit-Preference**: Prefer resolving with unit clauses. These reduce the size of resulting clauses.

- **Set-of-Support**: Divide the clauses into two sets:
  - A *known-satisfiable* subset
  - Other (called the “set of support” SOS)

- Always resolve with an SOS clause or a clause derived from an SOS clause.
Set-of-Support

- Showing that the following clause set is **unsatisfiable**:

\[
\{ \neg \phi_{11} \lor \neg \phi_{12} \lor \ldots \lor \neg \phi_{1m1} \lor \psi_1, \\
\neg \phi_{21} \lor \neg \phi_{22} \lor \ldots \lor \neg \phi_{1m2} \lor \psi_2, \\
\ldots \\
\neg \phi_{n1} \lor \neg \phi_{n2} \lor \ldots \lor \neg \phi_{nmn} \lor \psi_n, \\
\neg \chi_1 \lor \neg \chi_2 \lor \ldots \lor \neg \chi_p \}
\]

- Satisfiable “axioms”

- Initial set of support
Horn Clauses

- A Horn clause is one in which there is at most one non-negated literal:
  - $\neg \varphi_1 \lor \neg \varphi_2 \lor \ldots \lor \neg \varphi_m \lor \psi$ (one non-negated)
  - or
  - $\neg \varphi_1 \lor \neg \varphi_2 \lor \ldots \lor \neg \varphi_m$ (no non-negated)

- Horn clauses are the basis of the Prolog language, where:
  - $\neg \varphi_1 \lor \neg \varphi_2 \lor \ldots \lor \neg \varphi_m \lor \psi$
  - is written
  - $\psi :\neg \varphi_1, \varphi_2, \ldots \varphi_m$
  - interpreted as
  - $\varphi_1 \land \varphi_2 \land \ldots \land \varphi_m \rightarrow \psi$

If $m = 0$, then we just write $\psi$. 
Prolog uses a special case of resolution to do its work ("SLD" = \textit{Selective Linear Definite} resolution)

\begin{itemize}
  \item \{p \lor \neg r \lor \neg s,
  \quad r \lor \neg q,
  \quad s \lor \neg q,
  \quad q,
  \quad \neg p, \quad \text{Non-negated}
  \}
  \end{itemize}

becomes in Prolog syntax:

\begin{itemize}
  \item p :- r, s.
  \item r :- q.
  \item s :- q.
  \item q.
  \item \neg p, \quad \text{literals in red.}
  \end{itemize}

Dialog with Prolog:

\begin{verbatim}
consult(user).
p :- r, s.
r :- q.
s :- q.
q.
^D
l ?- p.
yes
\end{verbatim}
Resolution Theorem Provers

- Prolog cannot handle general negation
- Resolution theorem provers can
- Examples: Prover9, Vampire, ...
• Extends the former program “Otter”
• Developed at Argonne National Laboratory
• Free download for all platforms
• http://www.cs.unm.edu/~mccune/prover9/
• Also includes “Mace” for finding counterexamples
Prover9 GUI:  - is “not” 
| is “or”
Prover9 Proof ($F$ is empty clause)
Resolution for Predicate Logic

- **Predicate Clausal Form:**
  - A literal is an atomic formula or its negation (instead of a proposition symbol or its negation).

- The variables of each clause are each implicitly ∀-quantified.

- The variables of each clause are thus independent from the other clauses; even if they are the same, they should be thought of as being different (e.g. implicitly rename by indexing with a clause number).
Example: Predicate Clausal Form

- Clause set \{p(X), q(X, Y), ¬q(X, X) ∨ p(X)\}
  stands for the conjunction

- \( \forall X \ p(X) \)
  \( \land \forall X \forall Y \ q(X, Y) \)
  \( \land \forall X \forall Y (\neg q(X, X) \lor p(X)) \)

which is the same as

- \( \forall X_1 \ p(X_1) \)
  \( \land \forall X_2 \forall Y_2 \ q(X_2, Y_2) \)
  \( \land \forall X_3 \forall Y_3 (\neg q(X_3, X_3) \lor p(X_3)) \)

i.e. the clause set

- \{p(X_1), q(X_2, Y_2), ¬q(X_3, X_3) ∨ p(X_3)\}
How General is This?

- Completely general, as far as showing unsatisfiability is concerned.
Examples of Predicate Clausal Form

- ¬human(X) ∨ mortal(X)
- human(socrates)
- ¬mortal(socrates)

These clauses can be used to prove the syllogism:
- All humans are mortal.
- Socrates is a human.
- Therefore Socrates is mortal.
Resolution for Predicate Clauses

• To resolve *predicate* clauses, it is no longer sufficient to look for just a literal and its negation in two distinct clauses
  \[ \neg q(X, X) \lor p(X) \]
  \[ \neg p(Z) \lor r(Z, Y) \]

• For one thing, the identity of the *variables* is *independent* in each.

• For another, the arguments are generally *terms*, not just simple variables:
  \[ \neg q(X, X) \lor p(f(X)) \]
  \[ \neg p(X) \lor r(g(X), c) \]
Example of What Resolution Must Do

- Suppose we have derived three formulas (where c is a constant symbol):
  - $p(c)$
  - $\forall X (p(X) \rightarrow q(f(X)))$
  - $\forall X (q(X) \rightarrow r(X, g(X)))$

- We would expect to be able to infer
  - $q(f(c))$
  - $r(f(c), g(f(c)))$

- Resolution must be able to handle such things.
Equivalent Clausal Form

- The clausal form of
  - \( p(c) \)
  - \( \forall X \ (p(X) \rightarrow q(f(X))) \)
  - \( \forall X \ (q(X) \rightarrow r(X, g(X))) \)

  is
  - \{p(c), \neg p(X) \lor q(f(X)), \neg q(X) \lor r(X, g(X))\}

- Resolution has to “make a connection” between \( p(c) \) and \( p(X) \), and between \( q(f(X)) \) and \( q(X) \).
Unification

• The “connection” alluded to on the previous slide is known as unification.

• Two atoms are unifiable if there is a uniform set of substitutions of terms for their variables that makes them identical.

• If such a substitution set exists, it is applied to all literals in the formulas prior to resolution.
Unification Examples

- Consider atoms \( p(c), p(X) \) (\( c \) is a constant, \( X \) a variable).

- These terms are **unifiable**, since the substitution \([c/X]\) (substitute \( c \) for \( X \)) makes them identical.
Unification Examples

- Consider $q(c, d)$, $q(X, X)$ (c and d are constants, X a variable).

- These terms are **not unifiable**.

- Distinct **constant symbols do not unify**. There is no substitution that will make them identical.

- (Note: This is not the same as saying constant symbols cannot be equated. They can, with a separate equation such as $c = d$. **Equality is handled separately.**)
Renaming Apart

- Consider \( p(X, f(a)) \) vs. \( p(g(Y), f(X)) \)

- These might appear not to unify, since we would have a conflict \([g(Y)/X]\) vs. \([a/X]\).

- However, if we **rename** the variables in the second clause we get:
  \[
  p(X, f(a)) \text{ vs. } p(g(Z), f(W)).
  \]
  These unify, using \([g(Z)/X, a/W]\).

- **Note:** Renaming apart is done only at the **start** of considering unification of two clauses, and all variables in the clause are renamed **uniformly**.
Notation for Variable Substitutions

- In general, a substitution consists of a set of bindings of variables to terms, e.g.
  \[ \beta = [Z/X, f(Z, c)/Y, c/W] \]

- If \( \tau \) is a term, then \( \tau\beta \) denotes the result of making the substitutions \( \beta \) in for variables in \( \tau \), e.g.

\[
\begin{align*}
\text{if} & \quad \tau = p(X, g(Y, W)) \\
\text{then} & \quad \tau\beta = p(Z, g(f(Z, c), c))
\end{align*}
\]
Composing Variable Substitutions

• If $\beta$ and $\gamma$ are substitutions and $\tau$ is a term, then $(\tau\beta)\gamma$ denotes the result of first applying $\beta$ to $\tau$, then $\gamma$ to the result, e.g.

\[
\begin{align*}
\tau &= p(X, g(Y, W)) & \text{literal} \\
\beta &= \{Z/X, f(Z, c)/Y, c/W\} & \text{sub} \\
\gamma &= \{V/Z\} & \text{sub} \\
(\tau\beta)\gamma &= p(V, g(f(V, c), c))
\end{align*}
\]

• The \textbf{composition $\beta\gamma$ of substitutions} $\beta$ and $\gamma$ is the substitution such that for all terms $\tau$

\[
\tau(\beta\gamma) = (\tau\beta)\gamma
\]
e.g. $\{V/X, f(V, c)/Y, c/W\}$ above
Unifiers

- A set of substitutions that unifies two literals is called a **unifier**.
## More Unification Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifier, if any?</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(X, X)</td>
<td>p(f(Y), f(Z))</td>
<td></td>
</tr>
<tr>
<td>p(X, X)</td>
<td>p(f(Y), g(Y))</td>
<td></td>
</tr>
<tr>
<td>p(X, Y)</td>
<td>p(Z, f(Z))</td>
<td></td>
</tr>
<tr>
<td>p(X, f(X))</td>
<td>p(g(Y), W)</td>
<td></td>
</tr>
<tr>
<td>p(X, f(X))</td>
<td>p(f(Y), Y)</td>
<td></td>
</tr>
</tbody>
</table>
Even More Unification Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifier, if any?</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(X, Y)</td>
<td>p(f(Z), g(Z))</td>
<td></td>
</tr>
<tr>
<td>p(X, f(X))</td>
<td>p(f(Z), U)</td>
<td></td>
</tr>
<tr>
<td>p(f(X), g(X))</td>
<td>p(f(U), U)</td>
<td></td>
</tr>
<tr>
<td>p(f(X), f(X))</td>
<td>p(c, c)</td>
<td></td>
</tr>
<tr>
<td>p(f(X), g(X))</td>
<td>p(Y, g(Y))</td>
<td></td>
</tr>
</tbody>
</table>
Most General Unifiers (mgu)

- If two literals unify at all, they have a “most general unifier”, one which adds no unneeded constraints.

- Example: \( p(X) \) vs. \( p(f(Y)) \) could be unified with the substitution 
  \[ f(c)/X, \ c/Y \].

- However, this would **not** be the most general, since we could leave \( Y \) as a variable: 
  \[ f(Z)/X \]
  and each of the original literals would unify with this.
Generality of Substitutions

- Substitution $\beta$ is **as general as** substitution $\nu$ if there is a $\gamma$ such that $\nu = \beta \gamma$.

- To say that $\beta$ is a “most general unifier” means that it is as general as *any* unifier.
Find the MGU or indicate non-unifiable

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>MGU?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p(X, Y) )</td>
<td>( p(Z, Z) )</td>
<td></td>
</tr>
<tr>
<td>( p(X, c) )</td>
<td>( p(Y, Y) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(X), Y) )</td>
<td>( p(W, f(Z)) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(X), Y) )</td>
<td>( p(Z, Y) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(Z), g(X)) )</td>
<td>( p(Y, g(Y)) )</td>
<td></td>
</tr>
</tbody>
</table>
MGU Algorithm (Martelli & Montanari)

- **Input:** Two terms, or two atoms, $\tau_1$, $\tau_2$, already renamed apart.
- **Output:** Either the most general unifier for $\tau_1$, $\tau_2$, or “not unifiable”.

- $S := \{[\tau_1, \tau_2]\}$;  
  $\mu :=$ the empty substitution;  
  while( $S \neq \emptyset$ )  
  remove a pair $[L, R]$ from $S$;  
  if( $L = R$ )  
    do nothing;  
  else if( $L = f(s_1, s_2, ..., s_n)$ and $R = f(t_1, t_2, ..., t_n)$ )  
    $S := S \cup \{[s_1, t_1], [s_2, t_2],... [s_n, t_n]\}$;  
  else if( $L = x$ where $x$ is a variable not occurring in $R$)  
    $\mu := \mu[R/x]$;  
    $S :=$ applytoallpairs([R/x], S);  
  else if( $R = x$ where $x$ is a variable not occurring in $L$)  
    $\mu := \mu[L/x]$;  
    $S :=$ applytoallpairs([L/x], S);  
  else return “not unifiable”;  
  return $\mu$ as the MGU;

Intuitive Unification

- Remember when two things **don’t** unify:
  - Distinct constant symbols don’t unify.
  - Terms with outermost function symbols that are distinct don’t unify.
  - A term with an outermost function symbol doesn’t unify with a constant.
  - Two terms with the same outermost function symbol don’t unify if some of their arguments don’t pairwise unify.

- Remember that substitutions are **cumulative** during unification.
Example

- \(p(X, f(X)) \text{ vs. } p(Y, f(Y))\)  
  - Initial
  - \(S := \{[p(X, f(X)), p(Y, f(Y))]\}\)
  - \(\mu := []\)

- Remove \([p(X, f(X)), p(Y, f(Y))]\)
  - Case 2
  - \(S := \{[X, Y], [f(X), f(Y)]\}\)

- Remove \([X, Y]\)
  - Case 3
  - \(\mu := [Y/X]; S := \{[f(Y), f(Y)]\}\)

- Remove \([f(Y), f(Y)]\)
  - Case 1
  - \(S := {}\)

- Result: unifiable with mgu \([Y/X]\)
Diagrammatically

- $p(X, f(X))$  
  $\uparrow$ substitution $[Y/X]$  
  $\downarrow p(Y, f(Y))$

- $p(Y, f(Y))$
  $\downarrow$$\downarrow$$\downarrow$$\downarrow$  
  $\downarrow p(Y, f(Y))$
Example

• \( p(X, f(X)) \) vs. \( p(f(Y), Y) \)  
  initial
• \( S := \{[p(X, f(X)), p(f(Y), Y)]\} \)
• \( \mu := \{\} \)

• Remove \([p(X, f(X)), p(f(Y), Y)]\)  
  case 2
• \( S := \{[X, f(Y)], [f(X), Y]\} \)

• Remove \([X, f(Y)]\)  
  case 3
• \( \mu := [f(Y)/X]; S := \{[f(f(Y)), Y]\} \)

• Remove \([f(f(Y)), Y]\)  
  case 5
• Result: not unifiable
Diagrammatically

- $p(X, f(X))$
  - $\uparrow\downarrow$
  - substitution $[f(Y)/X]$
  - $p(f(Y), Y)$

- $p(f(Y), f(f(Y)))$
  - $\uparrow\downarrow$
  - occur check fails, not unifiable
  - $p(f(Y), Y)$
Example

- \( p(X, g(Z), X) \) vs. \( p(f(Y), Y, W) \)  
- \( S := \{[p(X, g(Z), X), p(f(Y), Y, W)]\} \)  
- \( \mu := {} \)  
  
Remove \( [p(X, g(Z), X), p(f(Y), Y, W)] \)  
- \( S := \{[X, f(Y)], [g(Z), Y], [X, W]\} \)  
- \( \mu := [f(Y)/X]; S := \{[g(Z), Y], [f(Y), W]\} \)  
- \( \mu := [f(g(Z))/X, g(Z)/Y]; S := \{[f(g(Z)), W]\} \)  
- \( \mu := [f(g(Z))/X, g(Z)/Y, f(g(Z))/W]; S := {} \)  
- Result: unifiable with  
  \( \text{mgu} \ [f(g(Z))/X, g(Z)/Y, f(g(Z))/W] \)
Diagrammatically

• \( p(X, g(Z), X) \) vs.
  \[
  \uparrow \\
  p(f(Y), Y, W)
  \]
  substitution \([f(Y)/X]\)

• \( p(f(Y), g(Z), f(Y)) \) vs.
  \[
  \uparrow \downarrow \\
  p(f(Y), Y, W)
  \]
  substitution \([g(Z)/Y, f(g(Z))/X]\)

• \( p(f(g(Z)), g(Z), f(g(Z))) \) vs.
  \[
  \uparrow \downarrow \\
  p(f(g(Z)), g(Z), W)
  \]
  substitution \([f(g(Z))/W, g(Z)/Y, f(g(Z))/X]\)

• \( p(f(g(Z)), g(Z), f(g(Z))) \) vs.
  \[
  \uparrow \downarrow \\
  p(f(g(Z)), g(Z), f(g(Z)))
  \]
  substitution \([f(g(Z))/W, g(Z)/Y, f(g(Z))/X]\)