H250: Honors Colloquium – Introduction to Computation
Resolution in Predicate Logic

Marius Minea
marius@cs.umass.edu
Resolution is an *inference rule* that produces a *new clause* from two clauses with *complementary literals* ($p$ and $\neg p$).

\[
\begin{array}{c}
p \lor A \\
\neg p \lor B \\
\hline
A \lor B
\end{array}
\]

"From clauses $p \lor A$ and $\neg p \lor B$ we derive clause $A \lor B$"

New clause = *resolvent* of the two clauses with respect to $p$

Example: \( \text{res}_p(p \lor q \lor \neg r, \neg p \lor s) = q \lor \neg r \lor s \)

*Resolution is a valid inference rule*:
any assignment making premises true also makes conclusion true

\[ \{p \lor A, \neg p \lor B\} \models A \lor B \]

*Corollary*: if $A \lor B$ is a contradiction, so is $(p \lor A) \land (\neg p \lor B)$
if resolution reaches contradiction, we started from a contradiction
Resolution For Predicates

In predicate logic, a literal is a (possibly negated) predicate:
not $p$ and $\neg p$, but $P(arg1)$ and $\neg P(arg2)$ (different args)

To derive a new clause from $A \lor P(arg1)$ and $B \lor \neg P(arg2)$ must bring args to common form.

Variables in clauses will be (implicitly) universally quantified can take any value $\Rightarrow$ can substitute with any terms

Is there a substitution bringing the arguments to a common form?
Ex. 1: $P(x, g(y))$ and $P(a, z)$
Ex. 2: $P(x, g(y))$ and $P(z, a)$

Ex. 1: $x \mapsto a, z \mapsto g(y)$ yields $P(a, g(y)), P(a, g(y)) \Rightarrow$ same

Ex. 2: can’t substitute constant $a$ with $g(y)$ ($a$ is not a variable)
$g$ is an arbitrary function, don’t know if $y$ exists with $g(y) = a$
Substitution and Term Unification

A substitution is a function that associates terms to variables
{\(x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\)}

Two terms can be unified if there is a substitution that makes them equal
\(f(x, g(y, z), t)\{x \mapsto h(z), y \mapsto h(b), t \mapsto u\} = f(h(z), g(h(b), z), u)\)

Unification Rules
A variable \(x\) may be unified with any term \(t\) (substitution) if \(x\) does not occur in \(t\) (otherwise, we'd get an infinite term) can't unify: \(x\) with \(f(h(y), g(x, z))\); but can trivially unify \(x\) with \(x\)

Two terms \(f(\ldots)\) can be unified if they have the same function \(f\) and the arguments (terms) can be unified one by one
⇒ two constants (0-arg functions): unified if equal
Implementing Unification: Union-Find

Unification defines equivalence classes:
If we unify $x$ with $y$ and then $y$ with $f(z, a)$,
then $x$ is also unified with $f(z, a) \Rightarrow$ equivalence

Must track equivalent variables.

*Union-Find*: data structure for building equivalence classes

Operations:
*find*(element): finds representative of equivalence class
*union*(elem1, elem2): makes elements equivalent (will stay so)
Union-Find Example

One implementation: set of trees with links up to parent

*find*: returns root of tree

*union*: links one root to the other

\[ g(Z) \]
\[ \uparrow \]
\[ Y \]
\[ \uparrow \]
\[ X \]

\[ \text{find}(X) = \text{find}(Y) = g(Z) \]

\[ g(Z) \]
\[ \uparrow \]
\[ Y \quad T \]
\[ \uparrow \]
\[ X \quad S \]

union\((Y, S)\) links \text{find}(S) with \text{find}(Y)

We maintain a *map* of variables to *terms*. 
Union-Find Example

Unify \( f(x, g(x, s(z)), t) \) with \( f(h(z), g(h(b), u), z) \)

\( x \) with \( h(z) \) \( \Rightarrow \) \{ \( x \mapsto h(z) \) \}

\( g(x, s(z)) \) with \( g(h(b), u) \) \( \Rightarrow \)
\( x \) with \( h(b) \) \( \Rightarrow \) \( h(z) \) with \( h(b) \) \( \Rightarrow \) \{ \( x \mapsto h(z), z \mapsto b \) \}
\( s(z) \) with \( u \) \( \Rightarrow \) \{ \( x \mapsto h(z), z \mapsto b, u \mapsto s(z) \) \}

\( t \) with \( z \) \( \Rightarrow \) \( t \) with \( b \) \( \Rightarrow \) \{ \( x \mapsto f(z), z \mapsto b, u \mapsto s(z), t \mapsto b \) \}

Substituting all the way:
\{ \( x \mapsto f(b), z \mapsto b, u \mapsto s(b), t \mapsto b \) \}

This substitution is the **most general unifier** of the given terms.
Resolution in Predicate Logic

Take clauses $A$ with $P(\ldots)$ (positive) and $B$, with $\neg P(\ldots)$ (negated)

$A$: $P(x, g(y)) \lor P(h(a), z) \lor Q(z)$  
$B$: $\neg P(h(z), t) \lor R(t, z)$

Choose some ($\geq 1$) $P(\ldots)$ from $A$ and some $\neg P(\ldots)$ from $B$

*Rename* common variables ($A$ and $B$ are independent clauses)

$A$: $P(x, g(y)) \lor P(h(a), z) \lor Q(z)$  
$B$: $\neg P(h(z_2), t) \lor R(t, z_2)$

*Unify all chosen* $P(\ldots)$ from $A$ and $\neg P(\ldots)$ from $B$

$$\{P(x, g(y)), P(h(a), z), P(h(z_2), t)\} \quad x \mapsto h(a); z_2 \mapsto a; z, t \mapsto g(y)$$

*Eliminate* chosen $P(\ldots)$ and $\neg P(\ldots)$ from $A \lor B$.

*Apply* resulting substitution and *add* new clause to list

$$Q(g(y)) \lor R(g(y), a)$$

*Keep* original clauses for unification with other predicates
We proceed similarly to CNF transformation, but with extra steps.

- rewrite $\rightarrow$, $\leftrightarrow$, etc., keep only $\land$, $\lor$, $\neg$
- push negation inwards to predicates (negation normal form)
- rename to get unique variable names (we’ll remove quantifiers)

\[
\forall x \ P(x) \lor \forall x \ \exists y \ Q(x, y) \quad \text{becomes} \quad \forall x \ P(x) \lor \forall z \ \exists y \ Q(z, y)
\]
We want to only keep universally quantified variables, and make quantification implicit.
⇒ use instantiation for existential quantifiers
In $\forall x_1 \ldots \forall x_k \exists y : (...)$, the choice of $y$ depends on $x_1, \ldots x_k$;
introduce a new Skolem function $y = f(x_1, \ldots, x_k)$, eliminating $y$
i.e., instantiate $y$ with $f(x_1, \ldots, x_k)$

In particular ($k = 0$), $\exists y$ outside any $\forall$ is instantiated with a new Skolem constant

*New* function or constant names for *each* existential quantifier.
Transformation Steps (cont’d.)

- Bring all $\forall$ quantifiers to the front (prenex normal form)
  $\forall x P(x) \land \forall y Q(y) \iff \forall x \forall y (P(x) \land Q(y))$
  $\forall x P(x) \lor \forall y Q(y) \iff \forall x \forall y (P(x) \lor Q(y))$

- Remove all quantifiers (implicit universal quantification)

- Bring $\land$ outside $\lor$ (convert to clausal form)
Exercise 9:

Every investor bought [something that is] stocks or bonds.
\[ A_1: \forall X (inv(X) \rightarrow \exists Y (buy(X, Y) \land (share(Y) \lor bond(Y)))) \]

If the Dow-Jones Average crashes, then all stocks that are not gold stocks fall.
\[ A_2: \text{crash} \rightarrow \forall X (share(X) \land \neg gold(X) \rightarrow falls(X)) \]

If the T-Bill interest rate rises, then all bonds fall.
\[ A_3: \text{tbrise} \rightarrow \forall X (\neg bond(X) \rightarrow falls(X)) \]

Every investor who bought something that falls is not happy.
\[ A_4: \forall X (inv(X) \rightarrow (\exists Y (buy(X, Y) \land falls(Y)) \rightarrow \neg happy(X))) \]
If the Dow-Jones Average crashes and the T-Bill interest rate rises, then any investor who is happy bought some gold stock.

\[ C: \text{crash} \land \text{tbrise} \rightarrow \forall X(\text{inv}(X) \land \text{happy}(X) \rightarrow \exists Y(\text{buy}(X, Y) \land \text{share}(Y) \land \text{gold}(Y))) \]

We negate the conclusion \textit{before} transforming quantifiers!

\[ \neg C: \neg (\text{crash} \land \text{tbrise} \rightarrow \forall X(\text{inv}(X) \land \text{happy}(X) \rightarrow \exists Y(\text{buy}(X, Y) \land \text{share}(Y) \land \text{gold}(Y)))) \]
Example in Negation Normal Form

\[ A_1: \forall X (\neg \text{inv}(X) \lor \exists Y (\text{buy}(X, Y) \land (\text{share}(Y) \lor \text{bond}(Y)))) \]
\[ A_2: \neg \text{crash} \lor \forall X (\neg \text{share}(X) \lor \text{gold}(X) \lor \text{falls}(X)) \]
\[ A_3: \neg \text{tbrise} \lor \forall X (\neg \text{bond}(X) \lor \text{falls}(X)) \]
\[ A_4: \forall X (\neg \text{inv}(X) \lor \forall Y (\neg \text{buy}(X, Y) \lor \neg \text{falls}(Y)) \lor \neg \text{happy}(X)) \]
\[ \neg C: \text{crash} \land \text{tbrise} \land \\
\exists X (\text{inv}(X) \land \text{happy}(X) \land \forall Y (\neg \text{buy}(X, Y) \lor \neg \text{share}(Y) \lor \neg \text{gold}(Y))) \]
Example: Skolemization

\[ A_1: \forall X (\neg \text{inv}(X) \lor \exists Y (\text{buy}(X, Y) \land (\text{share}(Y) \lor \text{bond}(Y)))) \]

Item \( Y \) bought depends on investor \( X \), \( Y = f(X) \)
\[ \forall X (\neg \text{inv}(X) \lor (\text{buy}(X, f(X)) \land (\text{share}(f(X)) \lor \text{bond}(f(X)))))) \]

\( X \) in \( \exists X (...) \) becomes constant \( b \)
\[ \neg C: \text{crash} \land \text{tbrise} \land \exists X (\text{inv}(X) \land \text{happy}(X)) \]
\[ \land \forall Y (\neg \text{buy}(X, Y) \lor \neg \text{share}(Y) \lor \neg \text{gold}(Y)) \]
\[ \text{crash} \land \text{tbrise} \land \text{inv}(b) \land \text{happy}(b) \]
\[ \land \forall C (\neg \text{buy}(b, Y) \lor \neg \text{share}(Y) \lor \neg \text{gold}(Y)) \]
Example: Eliminating Quantifiers

All remaining variables are arbitrary (implicit universal quantification)

$A_1$: $\neg \text{inv}(X) \lor (\text{buy}(X, f(X)) \land (\text{share}(f(X)) \lor \text{bond}(f(X))))$

$A_2$: $\neg \text{crash} \lor \neg \text{share}(X) \lor \text{gold}(X) \lor \text{falls}(X)$

$A_3$: $\neg \text{tbrise} \lor \neg \text{bond}(X) \lor \text{falls}(X)$

$A_4$: $\neg \text{inv}(X) \lor \neg \text{buy}(X, Y) \lor \neg \text{falls}(Y) \lor \neg \text{happy}(X)$

$\neg C$: $\text{crash} \land \text{tbrise} \land \text{inv}(b) \land \text{happy}(b)$

$\land (\neg \text{buy}(b, Y) \lor \neg \text{share}(Y) \lor \neg \text{gold}(Y))$

Next: apply distributivity of $\lor$ over $\land$ and write clauses separately
Example: Clausal Form

(1) \(\neg inv(X) \lor buy(X, f(X))\)
(2) \(\neg inv(X) \lor share(f(X)) \lor bond(f(X))\)
(3) \(\neg crash \lor \neg share(X) \lor gold(X) \lor falls(X)\)
(4) \(\neg tbrise \lor \neg bond(X) \lor falls(X)\)
(5) \(\neg inv(X) \lor \neg buy(X, Y) \lor \neg falls(Y) \lor \neg happy(X)\)
(6) \(crash\)
(7) \(tbrise\)
(8) \(inv(b)\)
(9) \(happy(b)\)
(10) \(\neg buy(b, Y) \lor \neg share(Y) \lor \neg gold(Y)\)
Proof: Generating Resolvents

(1) \( \neg \text{inv}(X) \lor \text{buy}(X, f(X)) \)
(2) \( \neg \text{inv}(X) \lor \text{share}(f(X)) \lor \text{bond}(f(X)) \)
(3) \( \neg \text{crash} \lor \neg \text{share}(X) \lor \text{gold}(X) \lor \text{falls}(X) \)
(4) \( \neg \text{tbrise} \lor \neg \text{bond}(X) \lor \text{falls}(X) \)
(5) \( \neg \text{inv}(X) \lor \neg \text{buy}(X, Y) \lor \neg \text{falls}(Y) \lor \neg \text{happy}(X) \)
(6) \text{crash}
(7) \text{tbrise}
(8) \text{inv}(b)
(9) \text{happy}(b)
(10) \( \neg \text{buy}(b, Y) \lor \neg \text{share}(Y) \lor \neg \text{gold}(Y) \)

Try to progressively eliminate predicates

(11) \( \neg \text{share}(X) \lor \text{gold}(X) \lor \text{falls}(X) \) \hspace{1cm} (3, 6)
(12) \( \neg \text{buy}(b, Y) \lor \neg \text{share}(Y) \lor \text{falls}(Y) \) \hspace{1cm} (10, 11, \ X = Y)
(13) \( \neg \text{bond}(X) \lor \text{falls}(X) \) \hspace{1cm} (4, 7)
Deriving Empty Clause

(1) $\neg inv(X) \lor buy(X, f(X))$
(2) $\neg inv(X) \lor share(f(X)) \lor bond(f(X)))$
(5) $\neg inv(X) \lor \neg buy(X, Y) \lor \neg falls(Y) \lor \neg happy(X)$
(8) $inv(b)$
(9) $happy(b)$
(12) $\neg buy(b, Y) \lor \neg share(Y) \lor falls(Y)$ \hspace{1cm} (10, 11, $X = Y$)
(13) $\neg bond(Y) \lor falls(Y)$ \hspace{1cm} rename for unification with (2)
(14) $\neg inv(X) \lor share(f(X)) \lor falls(f(X))$ \hspace{1cm} (2, 13, $Y = X$)
(15) $\neg buy(b, f(X)) \lor \neg inv(X) \lor falls(f(X))$ \hspace{1cm} (12, 14, $Y = f(X)$)
(16) $\neg buy(b, Y) \lor \neg falls(Y) \lor \neg happy(b)$ \hspace{1cm} (5, 8, $X = b$)
(17) $\neg buy(b, Y) \lor \neg falls(Y)$ \hspace{1cm} (9, 16)
(18) $\neg buy(b, f(X)) \lor \neg inv(X)$ \hspace{1cm} (15, 17, $Y = f(X))$
(19) $\neg inv(b)$ \hspace{1cm} (1, 18, $X = b$)
(20) $\emptyset$ (proof by contradiction done) \hspace{1cm} (8, 19)
Revisiting Resolution

We try to prove:

$$A_1 \land A_2 \land \ldots \land A_n \rightarrow C$$

by contradiction, negating the conclusion and showing

$$A_1 \land A_2 \land \ldots \land A_n \land \neg C$$

is a contradiction

We repeatedly generate new clauses (resolvents) by resolution with unification.

If we get the empty clause, the initial formula is unsatisfiable
If we can’t find new resolvents, the formula is satisfiable

Resolution in predicate logic is refutation-complete
for any unsatisfiable formula, we’ll get the empty clause
but can’t determine satisfiability of any formula
(there are formulas for which the procedure never stops)