Lecture 12

Unification. Resolution

Isabela Drămnesc UVT

Computer Science Department,
West University of Timișoara,
Romania

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.
- For practical purposes, given a finite set of ground formulae, one can rename the distinct ground atoms by distinct propositional formulae and thus answer the question of unsatisfiability by **propositional resolution**.
  See [Crăciun, 2010] for details on propositional resolution.

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.
- For practical purposes, given a finite set of ground formulae, one can rename the distinct ground atoms by distinct propositional formulae and thus answer the question of unsatisfiability by **propositional resolution**.
  See [Crăciun, 2010] for details on propositional resolution.
- However, this approach is not practical: there is no indication how to find the finite set of ground formulae: the set of possible ground instantiations is both unbounded and unstructured.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.

- An **expression** is a term or formula (in particular literal, clause, or set of clauses).

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.
- An **expression** is a term or formula (in particular literal, clause, or set of clauses).
- Let E be an expression, $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. An **instance of** E (or the result of applying θ to E), $E\theta$ is the expression obtained by **simultaneously** replacing every occurrence of x_i in E by t_i.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.
- An **expression** is a term or formula (in particular literal, clause, or set of clauses).
- Let E be an expression, $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. An **instance of** E (or the result of applying θ to E), $E\theta$ is the expression obtained by **simultaneously** replacing every occurrence of x_i in E by t_i.
- Example: let $E = p(x) \vee q(f(y))$, $\theta = \{x \leftarrow y, y \leftarrow f(a)\}$. Then

$$E\theta = p(y) \vee q(f(f(a))).$$

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ, $\theta\sigma$** is the substitution:

  $$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ,** $\theta\sigma$ is the substitution:

  $$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

  $$\theta = \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\},$$
  $$\sigma = \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\},$$

  then:

  $$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ,** $\theta\sigma$ is the substitution:

$$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

$$\theta = \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\},$$
$$\sigma = \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\},$$

  then:

$$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$

- Let E be an expression and θ, σ substitutions. Then $E(\theta\sigma) = (E\theta)\sigma$.

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ,** $\theta\sigma$ is the substitution:

$$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

$$\theta = \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\},$$
$$\sigma = \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\},$$

  then:

$$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$

- Let E be an expression and θ, σ substitutions. Then $E(\theta\sigma) = (E\theta)\sigma$.
- Let θ, σ, λ be substitutions. Then $\theta(\sigma\lambda) = (\theta\sigma)\lambda$.

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:
  - the substitution

    $$\{x \leftarrow f(a), y \leftarrow f(g(a)), z \leftarrow f(g(a))\}$$

    applied to both the literals will make them identical (will "unify" them),

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:
  - the substitution

$$\{x \leftarrow f(a), y \leftarrow f(g(a)), z \leftarrow f(g(a))\}$$

    applied to both the literals will make them identical (will "unify" them),
  - the same effect is obtained when applying the substitutions

$$\{x \leftarrow f(a), y \leftarrow a, z \leftarrow a\},$$

$$\{x \leftarrow f(a), z \leftarrow y\}.$$

- Given a set of literals, a **unifier** is a substitution that makes the atoms of the set identical. A **most general unifier (mgu)** is a unifier μ such that any other unifier θ can be obtained from μ by a further substitution λ such that $\theta = \mu\lambda$.

- Given a set of literals, a **unifier** is a substitution that makes the atoms of the set identical. A **most general unifier (mgu)** is a unifier μ such that any other unifier θ can be obtained from μ by a further substitution λ such that $\theta = \mu\lambda$.
- Note that not all literals are unifiable: if the predicate symbols are different, the literals cannot be unified. Also, consider the case of $p(x)$ and $p(f(x))$. Since the substitution of the variable x has to be done in the same time, the terms x and $f(x)$ cannot be made identical, and the unification will fail.

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

- A set of term equations is in **solved form** iff:

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,
  - each variable x_i that appears on the left hand side of an equation does not appear elsewhere in a set.

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,
  - each variable x_i that appears on the left hand side of an equation does not appear elsewhere in a set.

    A set of equations in solved form defines a substitution in a natural way by turning each equation $x_i = t_i$ into an element of the substitution, $x_i \leftarrow t_i$.

Unification Algorithm

INPUT: A set of term equations.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or “not unifiable”.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.
3. Let $t' = t''$ be an equation where t', t'' are not variables. If the outermost (function) symbol of t' and t'' are not identical, terminate and answer "not unifiable". Otherwise, if t' is of the form $f(t'_1, \ldots, t'_k)$ and t'' is of the form $f(t''_1, \ldots, t''_k)$, replace the equation $f(t'_1, \ldots, t'_k) = f(t''_1, \ldots, t''_k)$ by the k equations
$$t'_1 = t''_1, \ldots, t'_k = t''_k.$$

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.
3. Let $t' = t''$ be an equation where t', t'' are not variables. If the outermost (function) symbol of t' and t'' are not identical, terminate and answer "not unifiable". Otherwise, if t' is of the form $f(t'_1, \dots, t'_k)$ and t'' is of the form $f(t''_1, \dots, t''_k)$, replace the equation $f(t'_1, \dots, t'_k) = f(t''_1, \dots, t''_k)$ by the k equations
$$t'_1 = t''_1, \dots, t'_k = t''_k.$$
4. Let $x = t$ a term equation such that x has another occurrence in the set of term equations. If x occurs in t (occurs check!), terminate and answer "not unifiable". Otherwise, transform the equation set by replacing each occurrence of x in other equations by t.

Example (Unification, from [Ben-Ari, 2001])

Consider the following two equations:

$$g(y) = x$$
$$f(x, h(x), y) = f(g(z), w, z).$$

Example (Unification, from [Ben-Ari, 2001])

Consider the following two equations:

$$g(y) = x$$
$$f(x, h(x), y) = f(g(z), w, z).$$

- Apply rule 1 to the first equation and rule 3 to the second equation:

$$x = g(y)$$
$$x = g(z)$$
$$h(x) = w$$
$$y = z.$$

Example (Unification, from [Ben-Ari, 2001])

Consider the following two equations:

$$g(y) = x$$
$$f(x, h(x), y) = f(g(z), w, z).$$

- Apply rule 1 to the first equation and rule 3 to the second equation:

$$x = g(y)$$
$$x = g(z)$$
$$h(x) = w$$
$$y = z.$$

- Apply rule 4 on the second equation to replace the other occurrences of x:

$$g(z) = g(y)$$
$$x = g(z)$$
$$h(g(z)) = w$$
$$y = z.$$

Example (Unification, from [Ben-Ari, 2001])

- Apply rule 3 to the first equation

$$
\begin{aligned}
z &= y \\
x &= g(z) \\
h(g(z)) &= w \\
y &= z.
\end{aligned}
$$

Example (Unification, from [Ben-Ari, 2001])

- Apply rule 3 to the first equation

$$
\begin{aligned}
z &= y \\
x &= g(z) \\
h(g(z)) &= w \\
y &= z.
\end{aligned}
$$

- Apply rule 4 on the last equation to replace y by z in the first equation, then erase the resulting $z = z$ using rule 2:

$$
\begin{aligned}
x &= g(z) \\
h(g(z)) &= w \\
y &= z.
\end{aligned}
$$

Example (Unification, from [Ben-Ari, 2001])

- Apply rule 3 to the first equation

$$
\begin{aligned}
z &= y \\
x &= g(z) \\
h(g(z)) &= w \\
y &= z.
\end{aligned}
$$

- Apply rule 4 on the last equation to replace y by z in the first equation, then erase the resulting $z = z$ using rule 2:

$$
\begin{aligned}
x &= g(z) \\
h(g(z)) &= w \\
y &= z.
\end{aligned}
$$

- Transform the second equation by rule 1:

$$
\begin{aligned}
x &= g(z) \\
w &= h(g(z)) \\
y &= z.
\end{aligned}
$$

Example (Unification, from [Ben-Ari, 2001])

- The algorithm terminates successfully. The resulting substitution

$$\{x \leftarrow g(z), w \leftarrow h(g(z)), y \leftarrow z\}$$

is the most general unifier of the initial set of equations.

Theorem (Correctness of the unification algorithm)

The unification algorithm terminates. If the algorithm terminates with the answer "not unifiable", there is no unifier for the set of term equations. If it terminates successfully, the resulting set of equations is in solved form and it defines an mgu

$$\mu = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

of the set of equations

Proof.

See [Ben-Ari, 2001], pp. 158.

- Ground resolution was not practical.

- Ground resolution was not practical.
- It turns out that a practical version of resolution is possible, using unification.

- Ground resolution was not practical.
- It turns out that a practical version of resolution is possible, using unification.
- Recall the notions of literal, clause, clause sets introduced in the previous lecture.

- Ground resolution was not practical.
- It turns out that a practical version of resolution is possible, using unification.
- Recall the notions of literal, clause, clause sets introduced in the previous lecture.
- **Notation.** Let L be a literal. We denote with L^c the complementary literal (i.e. L and L^c are opposite, one is the negation of the other).

Definition (General resolution step)

Let C_1, C_2 be clauses *with no variables in common*. Let $L_1 \in C_1$ and $L_2 \in C_2$ be literals in the clauses such that L_1 and ${L_2}^c$ can be unified by a mgu σ. Then C_1 and C_2 are said to be **clashing clauses**, that **clash on the literals** L_1 and L_2, and **resolvent of** C_1 **and** C_2 is the clause:

$$Res(C_1, C_2) = (C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma).$$

Example (Resolvent of two clauses)

Example (Resolvent of two clauses)

Consider the clauses:

$$p(f(x), g(y)) \vee q(x, y) \qquad \neg p(f(f(a)), g(z)) \vee q(f(a), g(z))$$

Example (Resolvent of two clauses)

Consider the clauses:

$$p(f(x), g(y)) \vee q(x, y) \qquad \neg p(f(f(a)), g(z)) \vee q(f(a), g(z))$$

$L_1 = p(f(x), g(y))$ and ${L_2}^c = p(f(f(a)), g(z))$ can be unified with the mgu $\{x \leftarrow f(a), y \leftarrow z\}$ and the resolvent of the clauses is:

$$q(f(a), z) \vee q(f(a), g(z)).$$

Example (Resolvent of two clauses)

Consider the clauses:

$$p(f(x), g(y)) \lor q(x, y) \qquad \neg p(f(f(a)), g(z)) \lor q(f(a), g(z))$$

$L_1 = p(f(x), g(y))$ and ${L_2}^c = p(f(f(a)), g(z))$ can be unified with the mgu $\{x \leftarrow f(a), y \leftarrow z\}$ and the resolvent of the clauses is:

$$q(f(a), z) \lor q(f(a), g(z)).$$

Example (Resolvent of two clauses)

Consider the clauses:

$$p(f(x), g(y)) \vee q(x, y) \qquad \neg p(f(f(a)), g(z)) \vee q(f(a), g(z))$$

$L_1 = p(f(x), g(y))$ and ${L_2}^c = p(f(f(a)), g(z))$ can be unified with the mgu $\{x \leftarrow f(a), y \leftarrow z\}$ and the resolvent of the clauses is:

$$q(f(a), z) \vee q(f(a), g(z)).$$

- Note that the requirement for clauses to have no variables in common does not impose any real restrictions on the clause set. Remember that clauses are implicitly universally quantified, so changing the name of a variable does not change the meaning of the clause set.

General Resolution Procedure.

INPUT: A set of clauses S.

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat
  - **Choose** $C_1, C_2 \in S_i$ clashing clauses and let $C = Res(C_1, C_2)$.

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat
  - **Choose** $C_1, C_2 \in S_i$ clashing clauses and let $C = Res(C_1, C_2)$.
  - If $C = \emptyset$ terminate with the answer "not satisfiable".

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat
  - **Choose** $C_1, C_2 \in S_i$ clashing clauses and let $C = Res(C_1, C_2)$.
  - If $C = \emptyset$ terminate with the answer "not satisfiable".
  - Otherwise, $S_{i+1} = S_i \cup \{C\}$.

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat
  - **Choose** $C_1, C_2 \in S_i$ clashing clauses and let $C = Res(C_1, C_2)$.
  - If $C = \emptyset$ terminate with the answer "not satisfiable".
  - Otherwise, $S_{i+1} = S_i \cup \{C\}$.
- until $S_{i+1} = S_i$

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

General Resolution Procedure.

INPUT: A set of clauses S.

OUTPUT: S is satisfiable or S is not satisfiable. Also the algorithm may not terminate.

- Start with $S_0 = S$.
- Repeat
  - **Choose** $C_1, C_2 \in S_i$ clashing clauses and let $C = Res(C_1, C_2)$.
  - If $C = \emptyset$ terminate with the answer "not satisfiable".
  - Otherwise, $S_{i+1} = S_i \cup \{C\}$.
- until $S_{i+1} = S_i$
- Return "satisfiable".

Note that the algorithm above may not terminate, it is not a decision procedure (indeed that would not be expected since first order predicate logic is undecidable). The reason for nontermination is the existence of infinite models.

Example (Resolution refutation, from [Ben-Ari, 2001])

1.	$\neg p(x) \vee q(x) \vee r(x, f(x))$		
2.	$\neg p(x) \vee q(x) \vee s(f(x))$		
3.	$t(a)$		
4.	$p(a)$		
5.	$\neg r(a, y) \vee t(y)$		
6.	$\neg t(x) \vee \neg q(x)$		
7.	$\neg t(x) \vee \neg s(x)$		
8.	$\neg q(a)$	$x \leftarrow a$	3, 6
9.	$q(a) \vee s(f(a))$	$x \leftarrow a$	2.4
10.	$s(f(a))$		8, 9
11.	$q(a) \vee r(a, f(a))$	$x \leftarrow a$	1, 4
12.	$r(a, f(a))$		8.11
13.	$t(f(a))$	$y \leftarrow f(a)$	5, 12
14.	$\neg s(f(a))$	$x \leftarrow f(a)$	7, 13
15.	$\emptyset$		10, 14

Example (Resolution refutation with variable renaming, from [Ben-Ari, 2001])

First four clauses represent the initial clause set.

1.	$\neg p(x, y) \vee p(y, x)$	
2.	$\neg p(x, y) \vee \neg p(y, z) \vee p(x, z)$	
3.	$p(x, f(x))$	
4.	$p(x, x)$	
$3'$.	$p(x', f(x'))$ Rename 3.	
5.	$p(f(x), x)$	$\sigma_1 = \{y \leftarrow f(x), x' \leftarrow x\} 1, 3'$
$3''$.	$p(x'', f(x''))$ Rename 3	
6.	$\neg p(f(x), z) \vee p(x, z)$	$\sigma_2 = \{y \leftarrow f(x), x'' \leftarrow x\} 2, 3'$
$5'''$.	$p(f(x'''), x''')$ Rename 5	
7.	$p(x, x)$	$\sigma_3 = \{z \leftarrow x, x''' \leftarrow x\} 6, 5'''$
$4''''$.	$\neg p(x'''', x'''')$ Rename 4	
8.	$\emptyset$	$\sigma_4 = \{x'''' \leftarrow x\} 7, 4''''$

Resolution refutation with variable renaming, from [Ben-Ari, 2001].

- The substitution resulting from composing all intermediary substitutions:

$$\sigma = \sigma_1\sigma_2\sigma_3\sigma_4 =$$

$$\{y \leftarrow f(x), z \leftarrow x, x' \leftarrow x, x'' \leftarrow x, x''' \leftarrow x, x'''' \leftarrow x\}$$

- Restricted to the variables from the initial set, the resulting substitution is:

$$\sigma = \{y \leftarrow f(x), z \leftarrow x\}$$

Theorem (Soundness of substitution)

Theorem (Completeness of substitution)

If a set of clauses is unsatistiable, then the empty clause $\emptyset$ can be derived by the resolution procedure.

- For details on how the proofs of these theorems,
  see [Ben-Ari, 2001].

Theorem (Soundness of substitution)

If the unsatisfiable clause $\emptyset$ is derived during the general resolution procedure, then the set of clauses is unsatisfiable.

Theorem (Completeness of substitution)

If a set of clauses is unsatistiable, then the empty clause $\emptyset$ can be derived by the resolution procedure.

- For details on how the proofs of these theorems, see [Ben-Ari, 2001].

Some remarks on the resolution procedure

- Note that the resolution procedure is nondeterministic: which clashing clause to choose and which clashing literals to resolve on is not specified.

Some remarks on the resolution procedure

- Note that the resolution procedure is nondeterministic: which clashing clause to choose and which clashing literals to resolve on is not specified.
- Good choices will lead to the result quickly, while bad choices may lead to the algorithm not terminating.

Some remarks on the resolution procedure

- Note that the resolution procedure is nondeterministic: which clashing clause to choose and which clashing literals to resolve on is not specified.
- Good choices will lead to the result quickly, while bad choices may lead to the algorithm not terminating.
- The completeness theorem says that if the clause set is unsatisfiable a resolution refutation (generation of the empty clause) exists, i.e. that which uses good choices. Variants with bad choices may miss the solution.

- Read: Chapter 7, sections 7.5-7.8 of [Ben-Ari, 2001].
- Items of interest:
  - Ground resolution, impracticality of ground resolution (reasons).
  - Substitutions: compositions of substitutions, unifiers, most general unifiers.
  - Unification procedure.
  - General resolution, completeness of resolution (no proof).

- Read: Chapter 7, sections 7.1-7.4 of [Ben-Ari, 2001].
- Items of interest (no proofs required):
  - Predicate logic language: syntax, semantics(interpretation, model).
  - Herbrand's universe, Herbrand base, Herbrand interpretation
  - Herbrand's theorem, the significance of Herbrand's theorem.
  - Clausal form of first order formulae: Skolemization (Skolem constants, Skolem functions), transformation algorithm.

Ben-Ari, M. (2001).
Mathematical Logic for Computer Science.
Springer Verlag, London, 2nd edition.

Crăciun, A. (2005-2010).
Logic for Computer Science.