First-Order Logic
Resolution

Resolution for first-order logic

Gilmore's algorithm is correct and complete,
but useless in practice.

Resolution for first-order logic

Gilmore's algorithm is correct and complete,
but useless in practice.

We upgrade resolution to make it work for predicate logic.

Recall: resolution in propositional logic

Resolution step:

$$\begin{array}{ccc} \{L_1, \ldots, L_n, A\} & & \{L'_1, \ldots, L'_m, \neg A\} \\ & \{L_1, \ldots, L_n, L'_1, \ldots, L'_m\} & \end{array}$$

Recall: resolution in propositional logic

Resolution step:

$$
\begin{tikzcd}
\{L_1, \dots, L_n, A\} \arrow[dr, dash] & & \{L'_1, \dots, L'_m, \neg A\} \arrow[dl, dash] \\
& \{L_1, \dots, L_n, L'_1, \dots, L'_m\} &
\end{tikzcd}
$$

Resolution graph:

$$
\begin{tikzcd}
\{\neg A, B\} \arrow[dr, dash] & & \{A\} \arrow[dl, dash] & & \{\neg B\} \arrow[ddl, dash] \\
& \{B\} \arrow[drr, dash] & & & \\
& & & \square &
\end{tikzcd}
$$

Recall: resolution in propositional logic

Resolution step:

$$
\begin{tikzcd}[column sep=small]
\{L_1, \ldots, L_n, A\} \arrow[dr, dash] & & \{L'_1, \ldots, L'_m, \neg A\} \arrow[dl, dash] \\
& \{L_1, \ldots, L_n, L'_1, \ldots, L'_m\} &
\end{tikzcd}
$$

Resolution graph:

$$
\begin{tikzcd}[column sep=small]
\{\neg A, B\} \arrow[dr, dash] & & \{A\} \arrow[dl, dash] & & \{\neg B\} \arrow[ddl, dash] \\
& \{B\} \arrow[drr, dash] & & & \\
& & & \square &
\end{tikzcd}
$$

A set of clauses is unsatisfiable iff the empty clause can be derived.

Adapting Gilmore's Algorithm

Gilmore's Algorithm:

Let F be a closed formula in Skolem form
and let $F_1, F_2, F_3, \ldots$ be an enumeration of $E(F)$.

$n := 0$;
repeat $n := n + 1$
until $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$ is unsatisfiable;
  – *this can be checked with any calculus for propositional logic*
return "unsatisfiable"

Adapting Gilmore's Algorithm

Gilmore's Algorithm:

Let F be a closed formula in Skolem form
and let $F_1, F_2, F_3, \ldots$ be an enumeration of $E(F)$.

$n := 0$;
repeat $n := n + 1$
until $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$ is unsatisfiable;
– *this can be checked with any calculus for propositional logic*
return "unsatisfiable"

"any calculus" $\rightsquigarrow$ use resolution for the unsatisfiability test

Terminology

Terminology

Literal/clause/CNF is defined as for propositional logic
but with the atomic formulas of predicate logic.

A ground term/formula/etc is a term/formula/etc
that does not contain any variables.

An instance of a term/formula/etc
is the result of applying a substitution to a term/formula/etc.

A ground instance
is an instance that does not contain any variables.

Clause Herbrand expansion

Let $F = \forall y_1 \dots \forall y_n \, F^*$ be a closed formula in Skolem form with F^* in CNF, and let $C_1, \dots, C_m$ be the clauses of F^*.

The clause Herbrand expansion of F is the set of ground clauses

$$CE(F) = \bigcup_{i=1}^{m} \{ C_i[t_1/y_1] \dots [t_n/y_n] \mid t_1, \dots, t_n \in T(F) \}$$

Lemma

$CE(F)$ is unsatisfiable iff $E(F)$ is unsatisfiable.

Clause Herbrand expansion

Let $F = \forall y_1 \ldots \forall y_n \, F^*$ be a closed formula in Skolem form with F^* in CNF, and let $C_1, \ldots, C_m$ be the clauses of F^*.

The clause Herbrand expansion of F is the set of ground clauses

$$CE(F) = \bigcup_{i=1}^{m} \{ C_i[t_1/y_1] \ldots [t_n/y_n] \mid t_1, \ldots, t_n \in T(F) \}$$

Lemma
$CE(F)$ is unsatisfiable iff $E(F)$ is unsatisfiable.

Proof. Informally speaking, "$CE(F) \equiv E(F)$".

Ground resolution algorithm

Let F be a closed formula in Skolem form with F^* in CNF.

Let $C_1, C_2, C_3, \ldots$ be an enumeration of $CE(F)$.

$n := 0$;
$S := \emptyset$;
repeat
  $n := n + 1$;
  $S := S \cup \{C_n\}$;
until $S \vdash_{Res} \Box$

return "unsatisfiable"

Ground resolution algorithm

Note: For example, $CE(F)$ can be enumerated according to the size of the substitutions.

Let $F = \forall y_1 \ldots \forall y_n\, F^*$ and let $C_1, \ldots, C_m$ be the clauses of F^*. For every $s \geq 0$, define

$$\mathcal{C}_s = \bigcup_{i=1}^{m} \left\{ C_i[t_1/y_1]\ldots[t_n/y_n] \;\middle|\; \begin{array}{c} t_1, \ldots, t_n \in T(F) \\ \text{and} \\ |t_1| + \cdots + |t_n| = s \end{array} \right\}$$

Ground resolution algorithm

Note: For example, $CE(F)$ can be enumerated according to the size of the substitutions.

Let $F = \forall y_1 \ldots \forall y_n \, F^*$ and let $C_1, \ldots, C_m$ be the clauses of F^*. For every $s \geq 0$, define

$$\mathcal{C}_s = \bigcup_{i=1}^{m} \left\{ C_i[t_1/y_1] \ldots [t_n/y_n] \;\middle|\; \begin{array}{c} t_1, \ldots, t_n \in T(F) \\ \text{and} \\ |t_1| + \cdots + |t_n| = s \end{array} \right\}$$

$\mathcal{C}_s$ is finite for every $s \geq 0$ and $CE(F) = \bigcup_{s=0}^{\infty} \mathcal{C}_s$.

So $CE(F)$ can be enumerated by enumerating $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \cdots$.

Ground resolution algorithm

Note: For example, $CE(F)$ can be enumerated according to the size of the substitutions.

Let $F = \forall y_1 \ldots \forall y_n \, F^*$ and let $C_1, \ldots, C_m$ be the clauses of F^*. For every $s \geq 0$, define

$$\mathcal{C}_s = \bigcup_{i=1}^{m} \left\{ C_i[t_1/y_1]\ldots[t_n/y_n] \;\middle|\; \begin{array}{c} t_1, \ldots, t_n \in T(F) \\ \text{and} \\ |t_1| + \cdots + |t_n| = s \end{array} \right\}$$

$\mathcal{C}_s$ is finite for every $s \geq 0$ and $CE(F) = \bigcup_{s=0}^{\infty} \mathcal{C}_s$.

So $CE(F)$ can be enumerated by enumerating $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \cdots$.

Ground resolution algorithm

Note: For example, $CE(F)$ can be enumerated according to the size of the substitutions.

Let $F = \forall y_1 \ldots \forall y_n \, F^*$ and let $C_1, \ldots, C_m$ be the clauses of F^*. For every $s \geq 0$, define

$$\mathcal{C}_s = \bigcup_{i=1}^{m} \left\{ C_i[t_1/y_1] \ldots [t_n/y_n] \;\middle|\; \begin{array}{c} t_1, \ldots, t_n \in T(F) \\ \text{and} \\ |t_1| + \cdots + |t_n| = s \end{array} \right\}$$

$\mathcal{C}_s$ is finite for every $s \geq 0$ and $CE(F) = \bigcup_{s=0}^{\infty} \mathcal{C}_s$.

So $CE(F)$ can be enumerated by enumerating $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \cdots$.

Note: The search for $\square$ can be performed incrementally every time S is extended, keeping the clauses generated in previous steps.

Ground resolution theorem

The correctness of the ground resolution algorithm can be rephrased as follows:

Theorem

A formula $F = \forall y_1 \ldots \forall y_n \; F^*$ with F^* in CNF is unsatisfiable iff there is a sequence of ground clauses $C_1, \ldots, C_m = \Box$ such that for every $i = 1, \ldots, m$

Ground resolution theorem

The correctness of the ground resolution algorithm can be rephrased as follows:

Theorem

A formula $F = \forall y_1 \ldots \forall y_n\ F^*$ with F^* in CNF is unsatisfiable iff there is a sequence of ground clauses $C_1, \ldots, C_m = \Box$ such that for every $i = 1, \ldots, m$

- *either C_i is a ground instance of a clause $C \in F^*$, i.e. $C_i = C[t_1/y_1] \ldots [t_n/y_n]$ where $t_1, \ldots, t_n \in T(F)$,*
- *or C_i is a resolvent of two clauses C_a, C_b with $a < i$ and $b < i$*

Beyond ground resolution: Intuition

Blind enumeration of ground clauses is extremely inefficient

Beyond ground resolution: Intuition

Blind enumeration of ground clauses is extremely inefficient

$$F^* = \{\, \{P(x)\}\,,\, \{\neg P(f(g(b,y))), Q(y)\}\,,\, \{\neg Q(g(f(z),f(z)))\}\,\}.$$

The algorithm can derive $\Box$ from just three ground clauses:

$$\{P(f(g(b,g(f(a),f(a)))))\}$$
$$\{\neg P(f(g(b,g(f(a),f(a))))), Q(g(f(a),f(a)))\}$$
$$\{\neg Q(g(f(a),f(a)))\}$$

Blind enumeration will generate the third clause early on, but it will only generate the first two after many (many!) superfluous clauses.

Beyond ground resolution: Intuition

Better: guided search with “lazy” substitutions

Beyond ground resolution: Intuition

Better: guided search with "lazy" substitutions

$$F^* = \{\,\{P(x)\}\,,\,\{\neg P(f(g(b,y))), Q(y)\}\,,\,\{\neg Q(g(f(z),f(z)))\}\,\}$$

Beyond ground resolution: Intuition

Better: guided search with "lazy" substitutions

$$F^* = \{\, \{P(x)\}\,,\ \{\neg P(f(g(b,y))), Q(y)\}\,,\ \{\neg Q(g(f(z),f(z)))\}\,\}$$

When resolving the first two clauses, delay the choice of substitution for x.

Commit only to replacing x by

$$f(g(b, \text{whatever-}y\text{-will-be-later-replaced-by}))$$

Beyond ground resolution: Intuition

Better: guided search with "lazy" substitutions

$$F^* = \{\,\{P(x)\}\,,\,\{\neg P(f(g(b,y))), Q(y)\}\,,\,\{\neg Q(g(f(z), f(z)))\}\,\}$$

When resolving the first two clauses, delay the choice of substitution for x.

Commit only to replacing x by

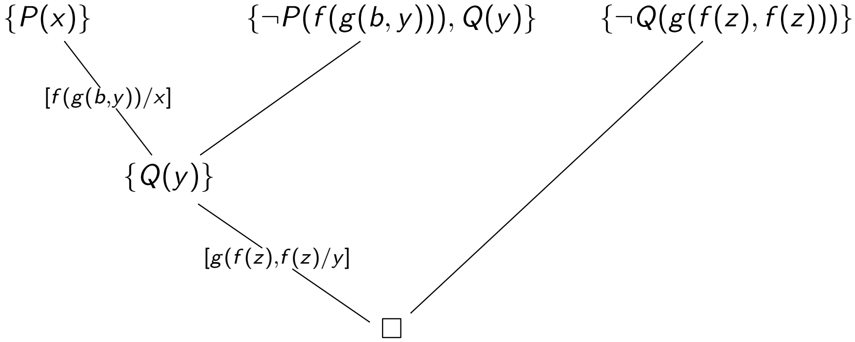
$$f(g(b, \text{whatever-}y\text{-will-be-later-replaced-by}))$$

For this:

- Allow substitutions with variables: $[f(g(b,y))/x]$.
- Apply substitutions only to two clauses that enable a new resolution step.

Beyond ground resolution: Intuition

$$F^* = \{\, \{P(x)\}\, , \, \{\neg P(f(g(b, y))), Q(y)\}\, , \, \{\neg Q(g(f(z), f(z)))\}\, \}$$



Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:
$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y.

Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:
$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y.

Example

$(P(x, y))[f(y)/x][b/y] \ =$

Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:
$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y.

Example

$(P(x,y))[f(y)/x][b/y] \; = \; (P(f(y),y))[b/y] \; =$

Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:
$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y.

Example

$(P(x,y))[f(y)/x][b/y] \;=\; (P(f(y),y))[b/y] \;=\; P(f(b),b)$

A composition of substitutions is again a substitution. $\sigma_1\sigma_2$ is the substitution that applies σ_1 first and then σ_2.

Substitutions as functions

Substitutions are functions from variables to terms:
$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:
$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y.

Example

$$(P(x, y))[f(y)/x][b/y] \;=\; (P(f(y), y))[b/y] \;=\; P(f(b), b)$$

A composition of substitutions is again a substitution. $\sigma_1\sigma_2$ is the substitution that applies σ_1 first and then σ_2.

Substitutions are functions. Therefore

$$\sigma_1 = \sigma_2 \quad \text{iff} \quad x\sigma_1 = x\sigma_2 \text{ for all variables } x$$

Substitutions as functions

Definition

The domain of a substitution σ is $dom(\sigma) = \{x \mid x\sigma \neq x\}$

Substitutions as functions

Definition

The domain of a substitution σ is $dom(\sigma) = \{x \mid x\sigma \neq x\}$

Example

$dom([a/x][b/y]) = \{x, y\}$

Substitutions as functions

Definition

The domain of a substitution σ is $dom(\sigma) = \{x \mid x\sigma \neq x\}$

Example

$dom([a/x][b/y]) = \{x, y\}$

Substitutions are defined to have finite domain, and so every substitution can be written as a

simultaneous substitution $[t_1/x_1, \ldots, t_n/x_n]$.

Unifier and most general unifier

Unifier and most general unifier

Let $\mathbf{L} = \{L_1, \ldots, L_k\}$ be a set of literals.

A substitution σ is a unifier of $\mathbf{L}$ if

$$L_1\sigma = L_2\sigma = \cdots = L_k\sigma$$

i.e. if $|\mathbf{L}\sigma| = 1$, where $\mathbf{L}\sigma = \{L_1\sigma, \ldots, L_k\sigma\}$.

$\mathbf{L}$ is unifiable if it has at least one unifier.

Unifier and most general unifier

Let $\mathbf{L} = \{L_1, \ldots, L_k\}$ be a set of literals.

A substitution σ is a unifier of $\mathbf{L}$ if

$$L_1\sigma = L_2\sigma = \cdots = L_k\sigma$$

i.e. if $|\mathbf{L}\sigma| = 1$, where $\mathbf{L}\sigma = \{L_1\sigma, \ldots, L_k\sigma\}$.

$\mathbf{L}$ is unifiable if it has at least one unifier.

A unifier σ of $\mathbf{L}$ is a most general unifier (mgu) of $\mathbf{L}$ if
for every unifier σ' of $\mathbf{L}$ there is a substitution δ such that $\sigma' = \sigma\delta$.

Unifier and most general unifier

Let $\mathbf{L} = \{L_1, \ldots, L_k\}$ be a set of literals.

A substitution σ is a unifier of $\mathbf{L}$ if

$$L_1\sigma = L_2\sigma = \cdots = L_k\sigma$$

i.e. if $|\mathbf{L}\sigma| = 1$, where $\mathbf{L}\sigma = \{L_1\sigma, \ldots, L_k\sigma\}$.

$\mathbf{L}$ is unifiable if it has at least one unifier.

A unifier σ of $\mathbf{L}$ is a most general unifier (mgu) of $\mathbf{L}$ if
for every unifier σ' of $\mathbf{L}$ there is a substitution δ such that $\sigma' = \sigma\delta$.

$$\cdot \xrightarrow{\sigma} \cdot$$

Unifier and most general unifier

Let $\mathbf{L} = \{L_1, \ldots, L_k\}$ be a set of literals.

A substitution σ is a unifier of $\mathbf{L}$ if

$$L_1\sigma = L_2\sigma = \cdots = L_k\sigma$$

i.e. if $|\mathbf{L}\sigma| = 1$, where $\mathbf{L}\sigma = \{L_1\sigma, \ldots, L_k\sigma\}$.

$\mathbf{L}$ is unifiable if it has at least one unifier.

A unifier σ of $\mathbf{L}$ is a most general unifier (mgu) of $\mathbf{L}$ if
for every unifier σ' of $\mathbf{L}$ there is a substitution δ such that $\sigma' = \sigma\delta$.

$$\begin{tikzcd} \cdot \arrow[r, "\sigma"] \arrow[dr, "\sigma'"] & \cdot \\ & \cdot \end{tikzcd}$$

Unifier and most general unifier

Let $\mathbf{L} = \{L_1, \ldots, L_k\}$ be a set of literals.

A substitution σ is a unifier of $\mathbf{L}$ if

$$L_1\sigma = L_2\sigma = \cdots = L_k\sigma$$

i.e. if $|\mathbf{L}\sigma| = 1$, where $\mathbf{L}\sigma = \{L_1\sigma, \ldots, L_k\sigma\}$.

$\mathbf{L}$ is unifiable if it has at least one unifier.

A unifier σ of $\mathbf{L}$ is a most general unifier (mgu) of $\mathbf{L}$ if
for every unifier σ' of $\mathbf{L}$ there is a substitution δ such that $\sigma' = \sigma\delta$.

$$\begin{tikzcd} \cdot \arrow[r, "\sigma"] \arrow[dr, "\sigma'"'] & \cdot \arrow[d, dashed, "\delta"] \\ & \cdot \end{tikzcd}$$

Exercise

Unifiable?			Yes	No
	$P(f(x))$	$P(g(y))$		
	$P(x)$	$P(f(y))$		
	$P(x)$	$P(f(x))$		
	$P(x, f(y))$	$P(f(u), f(z))$		
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|\mathbf{L}\sigma| > 1$ **do**

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|\mathbf{L}\sigma| > 1$ **do**

  Find the first position at which two literals $L_1, L_2 \in \mathbf{L}\sigma$ differ

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|\mathbf{L}\sigma| > 1$ **do**

Find the first position at which two literals $L_1, L_2 \in \mathbf{L}\sigma$ differ

if none of the two characters at that position is a variable

then return "non-unifiable"

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|\mathbf{L}\sigma| > 1$ **do**

  Find the first position at which two literals $L_1, L_2 \in \mathbf{L}\sigma$ differ

  if none of the two characters at that position is a variable

  then return "non-unifiable"

  else let x be the variable and t the term starting at that position

    if x occurs in t

    then return "non-unifiable"

    else $\sigma := \sigma\,[t/x]$

Unification algorithm

Input: a set $\mathbf{L} \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|\mathbf{L}\sigma| > 1$ **do**

Find the first position at which two literals $L_1, L_2 \in \mathbf{L}\sigma$ differ

if none of the two characters at that position is a variable

then return "non-unifiable"

else let x be the variable and t the term starting at that position

if x occurs in t

then return "non-unifiable"

else $\sigma := \sigma\,[t/x]$

return σ

Unification algorithm

Example

$\neg$`P(f(z,g(a,y)), h(z))`,

$\neg$`P(f(f(u,v),w), h(f(a,b)))`

Correctness of the unification algorithm

Lemma
The unification algorithm terminates.

Correctness of the unification algorithm

Lemma
The unification algorithm terminates.

Proof Every iteration of the **while**-loop (possibly except the last) replaces a variable x by a term t not containing x, and so the number of variables occurring in $\mathbf{L}\sigma$ decreases by one.

Correctness of the unification algorithm

Lemma

The unification algorithm terminates.

Proof Every iteration of the **while**-loop (possibly except the last) replaces a variable x by a term t not containing x, and so the number of variables occurring in $\mathbf{L}\sigma$ decreases by one.

Lemma

If $\mathbf{L}$ *is non-unifiable then the algorithm returns "non-unifiable".*

Correctness of the unification algorithm

Lemma

The unification algorithm terminates.

Proof Every iteration of the **while**-loop (possibly except the last) replaces a variable x by a term t not containing x, and so the number of variables occurring in $\mathbf{L}\sigma$ decreases by one.

Lemma

If $\mathbf{L}$ *is non-unifiable then the algorithm returns "non-unifiable".*

Proof If $\mathbf{L}$ is non-unifiable then the algorithm can never exit the loop normally.

Correctness/completeness of the unification algorithm

Lemma

If $\mathbf{L}$ *is unifiable then the algorithm returns the mgu of* $\mathbf{L}$
(and so in particular every unifiable set $\mathbf{L}$ *has an mgu).*

Correctness/completeness of the unification algorithm

Lemma

If **L** *is unifiable then the algorithm returns the mgu of* **L** *(and so in particular every unifiable set* **L** *has an mgu).*

Proof Assume **L** is unifiable and let n be the number of iterations of the loop on input **L**.

Correctness/completeness of the unification algorithm

Lemma

If $\mathbf{L}$ is unifiable then the algorithm returns the mgu of $\mathbf{L}$ (and so in particular every unifiable set $\mathbf{L}$ has an mgu).

Proof Assume $\mathbf{L}$ is unifiable and let n be the number of iterations of the loop on input $\mathbf{L}$.

Let $\sigma_0 = []$, for $1 \leq i \leq n$ let σ_i be the value of σ after the i-th iteration of the loop.

Correctness/completeness of the unification algorithm

Lemma

If $\mathbf{L}$ *is unifiable then the algorithm returns the mgu of* $\mathbf{L}$
(and so in particular every unifiable set $\mathbf{L}$ *has an mgu).*

Proof Assume $\mathbf{L}$ is unifiable and let n be the number of iterations of the loop on input $\mathbf{L}$.

Let $\sigma_0 = []$, for $1 \leq i \leq n$ let σ_i be the value of σ after the i-th iteration of the loop.

We prove for every $0 \leq i \leq n$:

(a) If $1 \leq i$, the i-th iteration does not return "non-unifiable".

(b) For every unifier σ' of $\mathbf{L}$ there is a substitution δ_i such that $\sigma' = \sigma_i \, \delta_i$.

Correctness/completeness of the unification algorithm

Lemma

If $\mathbf{L}$ *is unifiable then the algorithm returns the mgu of* $\mathbf{L}$
(and so in particular every unifiable set $\mathbf{L}$ *has an mgu).*

Proof Assume $\mathbf{L}$ is unifiable and let n be the number of iterations of the loop on input $\mathbf{L}$.

Let $\sigma_0 = []$, for $1 \leq i \leq n$ let σ_i be the value of σ after the i-th iteration of the loop.

We prove for every $0 \leq i \leq n$:

(a) If $1 \leq i$, the i-th iteration does not return "non-unifiable".

(b) For every unifier σ' of $\mathbf{L}$ there is a substitution δ_i such that $\sigma' = \sigma_i \, \delta_i$.

By (a) the algorithm exits the loop normally after n iterations.
By (b) it returns a most general unifier.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$):

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.
Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.
Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$\sigma_{i+1}\,\delta_{i+1}$

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$$\begin{aligned} & \sigma_{i+1}\,\delta_{i+1} \\ = \;& \sigma_i\,[t/x]\,\delta_{i+1} && \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \end{aligned}$$

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$$
\begin{array}{lll}
 & \sigma_{i+1}\,\delta_{i+1} & \\
= & \sigma_i\,[t/x]\,\delta_{i+1} & \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \\
= & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, t\delta_{i+1}/x] &
\end{array}
$$

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$$\begin{array}{lll} & \sigma_{i+1}\,\delta_{i+1} & \\ = & \sigma_i\,[t/x]\,\delta_{i+1} & \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \\ = & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, t\delta_{i+1}/x] & \\ = & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, u/x] & (u = t\delta_{i+1} \text{ by note}) \end{array}$$

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$$
\begin{array}{cll}
 & \sigma_{i+1}\,\delta_{i+1} & \\
= & \sigma_i\,[t/x]\,\delta_{i+1} & \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \\
= & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, t\delta_{i+1}/x] & \\
= & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, u/x] & (u = t\delta_{i+1} \text{ by note}) \\
= & \sigma_i\,\delta_i &
\end{array}
$$

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i:

Basis ($i = 0$): For (a) there is nothing to prove.
For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|\mathbf{L}\sigma_i| > 1$ and $\mathbf{L}\sigma_i$ unifiable, x and t exist and x does not occur in t, and so "non-unifiable" is not returned.

For (b): Let σ' be a unifier of $\mathbf{L}$. IH: $\sigma' = \sigma_i\delta_i$ for some δ_i.
δ_i must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$ where $x_1, \ldots, x_k, x$ are distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$.

Note: $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i\delta_i$ is unifier (IH), x not in t)

$$
\begin{array}{cll}
 & \sigma_{i+1}\,\delta_{i+1} & \\
= & \sigma_i\,[t/x]\,\delta_{i+1} & \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \\
= & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, t\delta_{i+1}/x] & \\
= & \sigma_i\,[t_1/x_1, \ldots, t_k/x_k, u/x] & (u = t\delta_{i+1} \text{ by note}) \\
= & \sigma_i\,\delta_i & \\
= & \sigma' & \text{(IH)}
\end{array}
$$

Renaming

Renaming

Definition

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

Resolvents for first-order logic

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

A clause R is a resolvent of two clauses C_1 and C_2 iff:

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

A clause R is a resolvent of two clauses C_1 and C_2 iff:

- there is a renaming ρ such that
  no variable occurs in both C_1 and $C_2\,\rho$ and
  ρ is injective on the set of variables in C_2;

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

A clause R is a resolvent of two clauses C_1 and C_2 iff:

- there is a renaming ρ such that
  no variable occurs in both C_1 and $C_2\,\rho$ and
  ρ is injective on the set of variables in C_2;
- there are literals $L_1, \ldots, L_m \in C_1$ $(m \geq 1)$
  and $L'_1, \ldots, L'_n \in C_2\,\rho$ $(n \geq 1)$ such that

  $$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

  is unifiable; and

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

A clause R is a resolvent of two clauses C_1 and C_2 iff:

- there is a renaming ρ such that
  no variable occurs in both C_1 and $C_2\,\rho$ and
  ρ is injective on the set of variables in C_2;
- there are literals $L_1, \ldots, L_m \in C_1 \quad (m \geq 1)$
  and $L'_1, \ldots, L'_n \in C_2\,\rho \quad (n \geq 1)$ such that

  $$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

  is unifiable; and
- $R = ((C_1 - \{L_1, \ldots, L_m\}) \cup (C_2\,\rho - \{L'_1, \ldots, L'_n\}))\,\sigma$
  for any mgu σ.

Resolvents for first-order logic

A substitution ρ is a renaming if for every variable x, $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

A clause R is a resolvent of two clauses C_1 and C_2 iff:

- there is a renaming ρ such that
  no variable occurs in both C_1 and $C_2\,\rho$ and
  ρ is injective on the set of variables in C_2;
- there are literals $L_1, \ldots, L_m \in C_1 \quad (m \geq 1)$
  and $L'_1, \ldots, L'_n \in C_2\,\rho \quad (n \geq 1)$ such that

  $$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

  is unifiable; and
- $R = ((C_1 - \{L_1, \ldots, L_m\}) \cup (C_2\,\rho - \{L'_1, \ldots, L'_n\}))\,\sigma$
  for any mgu σ.

Example

$C_1 = \{\ P(x),\ Q(x),\ P(g(y))\ \}$ and $C_2 = \{\ \neg P(x),\ R(f(x), a)\ \}$

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	1

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	0

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	0
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y, z)\}$	

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	0
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y, z)\}$	2

Why renaming?

Example

$\forall x(P(x) \land \neg P(f(x)))$

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps,

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$ such that for every C_i

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$ such that for every C_i

- either $C_i \in F$

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$ such that for every C_i

- either $C_i \in F$
- or C_i is the resolvent of C_a and C_b where $a, b < i$.

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$ such that for every C_i

- either $C_i \in F$
- or C_i is the resolvent of C_a and C_b where $a, b < i$.

Questions:

Correctness Does $F \vdash_{Res} \Box$ imply that F is unsatisfiable?

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \dots, C_m = C$ such that for every C_i

- either $C_i \in F$
- or C_i is the resolvent of C_a and C_b where $a, b < i$.

Questions:

Correctness Does $F \vdash_{Res} \Box$ imply that F is unsatisfiable?

Completeness Does unsatisfiability of F imply $F \vdash_{Res} \Box$?

Exercise

Derive $\square$ from the following clauses:

1. $\{\neg P(x), Q(x), R(x, f(x))\}$
2. $\{\neg P(x), Q(x), S(f(x))\}$
3. $\{T(a)\}$
4. $\{P(a)\}$
5. $\{\neg R(a, z), T(z)\}$
6. $\{\neg T(x), \neg Q(x)\}$
7. $\{\neg T(y), \neg S(y)\}$

Correctness of Resolution for First-Order Logic

Definition

The universal closure of a formula H with free variables $x_1, \ldots, x_n$:

$$\forall H \;=\; \forall x_1 \forall x_2 \ldots \forall x_n H$$

Correctness of Resolution for First-Order Logic

Definition

The universal closure of a formula H with free variables $x_1, \dots, x_n$:

$$\forall H \;=\; \forall x_1 \forall x_2 \dots \forall x_n H$$

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \ldots, C_m$ be the sequence of clauses leading to $\Box$.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \ldots, C_m$ be the sequence of clauses leading to $\Box$.
We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \dots, C_m$ be the sequence of clauses leading to $\Box$.
We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$.
Let C_m be a resolvent of C_a and C_b $(a, b < m)$. We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \ldots, C_m$ be the sequence of clauses leading to $\Box$.
We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$.
Let C_m be a resolvent of C_a and C_b $(a, b < m)$. We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of $(*)$: Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$. $\qquad (**)$
We prove $\mathcal{A}(\forall C_m) = 1$ by contradiction. Assume $\mathcal{A}(\forall C_m) = 0$.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \ldots, C_m$ be the sequence of clauses leading to $\Box$. We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$. Let C_m be a resolvent of C_a and C_b $(a, b < m)$. We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of $(*)$: Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$. $\qquad (**)$

We prove $\mathcal{A}(\forall C_m) = 1$ by contradiction. Assume $\mathcal{A}(\forall C_m) = 0$.

By def. $$\begin{aligned} C_m &= ((C_a - \{L_1, \ldots\}) \cup (C_b\rho - \{L'_1, \ldots\}))\sigma \\ &= (C_a\sigma - \{L\}) \cup (C_b\rho\sigma - \{\overline{L}\}) \end{aligned}$$

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \dots, C_m$ be the sequence of clauses leading to $\Box$.
We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$.
Let C_m be a resolvent of C_a and C_b ($a, b < m$). We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of $(*)$: Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$. $\qquad (**)$
We prove $\mathcal{A}(\forall C_m) = 1$ by contradiction. Assume $\mathcal{A}(\forall C_m) = 0$.
By def.

$$\begin{aligned} C_m &= ((C_a - \{L_1, \dots\}) \cup (C_b\rho - \{L'_1, \dots\}))\sigma \\ &= (C_a\sigma - \{L\}) \cup (C_b\rho\sigma - \{\overline{L}\}) \end{aligned}$$

$\Rightarrow \mathcal{A}'(C_m) = 0$ where $\mathcal{A}' = \mathcal{A}[u_1/x_1, \dots]$ for some $u_i \in U_{\mathcal{A}}$

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \dots, C_m$ be the sequence of clauses leading to $\Box$. We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$. Let C_m be a resolvent of C_a and C_b $(a, b < m)$. We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of $(*)$: Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$. $\qquad (**)$
We prove $\mathcal{A}(\forall C_m) = 1$ by contradiction. Assume $\mathcal{A}(\forall C_m) = 0$.
By def.

$$\begin{aligned} C_m &= ((C_a - \{L_1, \dots\}) \cup (C_b\rho - \{L'_1, \dots\}))\sigma \\ &= (C_a\sigma - \{L\}) \cup (C_b\rho\sigma - \{\overline{L}\}) \end{aligned}$$

$\Rightarrow \mathcal{A}'(C_m) = 0$ where $\mathcal{A}' = \mathcal{A}[u_1/x_1, \dots]$ for some $u_i \in U_{\mathcal{A}}$
$\Rightarrow \mathcal{A}'(C_a\sigma - \{L\}) = \mathcal{A}'(C_b\rho\sigma - \{\overline{L}\}) = 0$

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If $F^* \vdash_{Res} \Box$ then F is unsatisfiable.

Proof Let $C_1, \ldots, C_m$ be the sequence of clauses leading to $\Box$.
We prove $\forall F^* \models \forall C_m$ by induction on m. Trivial if $C_m \in F^*$.
Let C_m be a resolvent of C_a and C_b $(a, b < m)$. We prove

$$\forall C_a, \forall C_b \models \forall C_m \qquad (*)$$

Thus $\forall F^* \models \forall C_m$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of $(*)$: Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$. $\qquad (**)$
We prove $\mathcal{A}(\forall C_m) = 1$ by contradiction. Assume $\mathcal{A}(\forall C_m) = 0$.
By def.

$$\begin{aligned} C_m &= ((C_a - \{L_1, \ldots\}) \cup (C_b\rho - \{L'_1, \ldots\}))\sigma \\ &= (C_a\sigma - \{L\}) \cup (C_b\rho\sigma - \{\overline{L}\}) \end{aligned}$$

$\Rightarrow \mathcal{A}'(C_m) = 0$ where $\mathcal{A}' = \mathcal{A}[u_1/x_1, \ldots]$ for some $u_i \in U_{\mathcal{A}}$
$\Rightarrow \mathcal{A}'(C_a\sigma - \{L\}) = \mathcal{A}'(C_b\rho\sigma - \{\overline{L}\}) = 0$
$\Rightarrow \mathcal{A}'(L) = \mathcal{A}'(\overline{L}) = 1$ becs. $\mathcal{A}'(C_a\sigma) = \mathcal{A}'(C_b\rho\sigma) = 1$ becs. $(**)$
Contradiction

Completeness: The idea

Simulate ground resolution because that is complete

Completeness: The idea

Simulate ground resolution because that is complete

Lift the resolution proof from the ground resolution proof

Lifting Lemma

Lifting Lemma

C_1 C_2

Let C_1, C_2 be two clauses

Lifting Lemma

Let C_1, C_2 be two clauses and
let C_1', C_2' be two ground instances

$$\begin{array}{ccc} C_1 & \qquad & C_2 \\ \downarrow & & \downarrow \\ C_1' & & C_2' \end{array}$$

$\rightarrow$: Substitution

Lifting Lemma

Let C_1, C_2 be two clauses and let C_1', C_2' be two ground instances with (propositional) resolvent R'.

$$
\begin{tikzcd}
C_1 \arrow[d] & & C_2 \arrow[d] \\
C_1' \arrow[dr, dash] & & C_2' \arrow[dl, dash] \\
& R' &
\end{tikzcd}
$$

$\rightarrow$: Substitution

—: Resolution

Lifting Lemma

Let C_1, C_2 be two clauses and
let C_1', C_2' be two ground instances
with (propositional) resolvent R'.

Then there is a resolvent R of C_1, C_2

$$\begin{tikzcd}
C_1 \arrow[d] \arrow[dr, dash, dashed] & & C_2 \arrow[d] \arrow[dl, dash, dashed] \\
C_1' \arrow[dr, dash] & R & C_2' \arrow[dl, dash] \\
& R' &
\end{tikzcd}$$

$\rightarrow$: Substitution
—: Resolution

Lifting Lemma

Let C_1, C_2 be two clauses and
let C_1', C_2' be two ground instances
with (propositional) resolvent R'.

Then there is a resolvent R of C_1, C_2
such that R' is a ground instance of R.

$$
\begin{tikzcd}
C_1 \arrow[d] \arrow[dr, dash, dashed] & & C_2 \arrow[d] \arrow[dl, dash, dashed] \\
C_1' \arrow[dr, dash] & R \arrow[d, dashed] & C_2' \arrow[dl, dash] \\
& R' &
\end{tikzcd}
$$

$\rightarrow$: Substitution
—: Resolution

Lifting Lemma: example

$\{\neg P(f(x)), Q(x)\}$ $\{P(f(g(y)))\}$

Lifting Lemma: example

$$\{\neg P(f(x)), Q(x)\} \qquad\qquad \{P(f(g(y)))\}$$

$$\Big\downarrow [g(a)/x] \qquad\qquad\qquad \Big\downarrow [a/y]$$

Lifting Lemma: example

$$\begin{array}{c} \{\neg P(f(x)), Q(x)\} \\ \Big\downarrow [g(a)/x] \\ \{\neg P(f(g(a))), Q(g(a))\} \end{array} \qquad \begin{array}{c} \{P(f(g(y)))\} \\ \Big\downarrow [a/y] \\ \{P(f(g(a)))\} \end{array}$$

Lifting Lemma: example

$$
\begin{array}{ccccc}
\{\neg P(f(x)), Q(x)\} & & & & \{P(f(g(y)))\} \\
\downarrow [g(a)/x] & & & & \downarrow [a/y] \\
\{\neg P(f(g(a))), Q(g(a))\} & & & & \{P(f(g(a)))\} \\
& \searrow & & \swarrow & \\
& & \{Q(g(a))\} & &
\end{array}
$$

Lifting Lemma: example

$$
\begin{tikzcd}
\{\neg P(f(x)), Q(x)\} \arrow[d, "{[g(a)/x]}"] \arrow[dr, dash] & & \{P(f(g(y)))\} \arrow[d, "{[a/y]}"] \arrow[dl, dash] \\
\{\neg P(f(g(a))), Q(g(a))\} \arrow[dr, dash] & \{Q(g(y))\} & \{P(f(g(a)))\} \arrow[dl, dash] \\
& \{Q(g(a))\} &
\end{tikzcd}
$$

Lifting Lemma: example

$$\begin{tikzcd}
\{\neg P(f(x)), Q(x)\} \arrow[d, "{[g(a)/x]}"] \arrow[dr, dash] & & \{P(f(g(y)))\} \arrow[d, "{[a/y]}"] \arrow[dl, dash] \\
\{\neg P(f(g(a))), Q(g(a))\} \arrow[dr, dash] & \{Q(g(y))\} \arrow[d, "{[a/y]}"] & \{P(f(g(a)))\} \arrow[dl, dash] \\
& \{Q(g(a))\} &
\end{tikzcd}$$

Proof of Lifting Lemma.

Proof of Lifting Lemma.
(1) C_1', C_2' are ground instances of C_1, C_2

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

Proof of Lifting Lemma.
(1) C_1', C_2' are ground instances of C_1, C_2
(2) R' is propositional resolvent of C_1' and C_2'
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$.

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$

Proof of Lifting Lemma.

(1) C'_1, C'_2 are ground instances of C_1, C_2
(2) R' is propositional resolvent of C'_1 and C'_2
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables
(1) $\Rightarrow$ C'_2 is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.
$C'_1 = C_1\sigma_1$ and $C'_2 = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

Proof of Lifting Lemma.
(1) C_1', C_2' are ground instances of C_1, C_2
(2) R' is propositional resolvent of C_1' and C_2'
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables
(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.
$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$
$\Rightarrow$ $C_1' = C_1\sigma$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$

Proof of Lifting Lemma.
(1) C_1', C_2' are ground instances of C_1, C_2
(2) R' is propositional resolvent of C_1' and C_2'
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables
(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.
$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$
$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \ldots\} \subseteq C_1$ and $\{L_1', \ldots\} \subseteq C_2\rho$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\}$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2
(2) R' is propositional resolvent of C_1' and C_2'
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables
(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.
$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$
$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$
(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$
$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$
s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.
Let σ_0 be an mgu of M

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2
(2) R' is propositional resolvent of C_1' and C_2'
We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables
(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.
$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$
$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$
(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$
$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$
s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.
Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ
$\Rightarrow$ A resolvent of C_1 and C_2:
$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

$R\delta$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

$R\delta = ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

$R\delta = ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma$

$= (C_1\sigma - \{L\}) \cup (C_2\rho\sigma - \{\overline{L}\})$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

$R\delta = ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma$

$= (C_1\sigma - \{L\}) \cup (C_2\rho\sigma - \{\overline{L}\})$

$= (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$

Proof of Lifting Lemma.

(1) C_1', C_2' are ground instances of C_1, C_2

(2) R' is propositional resolvent of C_1' and C_2'

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow$ C_2' is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C_1' = C_1\sigma_1$ and $C_2' = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow$ $C_1' = C_1\sigma$ and $C_2' = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow$ $R' = (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$ where $L \in C_1'$ and $\overline{L} \in C_2'$

$\Rightarrow$ there are $\{L_1, \dots\} \subseteq C_1$ and $\{L_1', \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\overline{L_1}, \dots, L_1', \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

$\Rightarrow$ A resolvent of C_1 and C_2:

$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma_0$

$R\delta = ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L_1', \dots\}))\sigma$
$= (C_1\sigma - \{L\}) \cup (C_2\rho\sigma - \{\overline{L}\})$
$= (C_1' - \{L\}) \cup (C_2' - \{\overline{L}\})$
$= R'$

Completeness of Resolution for First-Order Logic

Theorem
Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$.

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \dots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \dots, C_n = \Box$

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C_1', \ldots, C_n' = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C_i' is a ground instance of C_i.

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C'_i is a ground instance of C_i.

If C'_i is a ground instance of some clause $C \in F^*$:

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C_1', \ldots, C_n' = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C_i' is a ground instance of C_i.

If C_i' is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C'_i is a ground instance of C_i.

If C'_i is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

If C'_i is a resolvent of C'_a, C'_b $(a, b < i)$:

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C_1', \ldots, C_n' = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C_i' is a ground instance of C_i.

If C_i' is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

If C_i' is a resolvent of C_a', C_b' $(a, b < i)$:
C_a', C_b' have been transformed already into C_a, C_b s.t. C_a', C_b' are ground instances of C_a, C_b.

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C'_i is a ground instance of C_i.

If C'_i is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

If C'_i is a resolvent of C'_a, C'_b $(a, b < i)$:
C'_a, C'_b have been transformed already into C_a, C_b s.t. C'_a, C'_b are ground instances of C_a, C_b. By the Lifting Lemma there is a resolvent R of C_a, C_b s.t. C'_i is a ground instance of R.

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF. If F is unsatisfiable then $F^* \vdash_{Res} \Box$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that C'_i is a ground instance of C_i.

If C'_i is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

If C'_i is a resolvent of C'_a, C'_b $(a, b < i)$:
C'_a, C'_b have been transformed already into C_a, C_b s.t. C'_a, C'_b are ground instances of C_a, C_b. By the Lifting Lemma there is a resolvent R of C_a, C_b s.t. C'_i is a ground instance of R.
Set $C_i = R$.

Resolution Theorem for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
Then F is unsatisfiable iff $F^* \vdash_{Res} \square$.

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\square \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
  there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
  such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\square \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
  there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
  such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$
  $\Rightarrow F$ is unsatisfiable

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
  there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
  such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$
  $\Rightarrow F$ is unsatisfiable
- The algorithm terminates and $\Box \notin S$

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$
  $\Rightarrow F$ is unsatisfiable
- The algorithm terminates and $\Box \notin S$
  $\Rightarrow F$ is satisfiable

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
  there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
  such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
 every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$
  $\Rightarrow F$ is unsatisfiable
- The algorithm terminates and $\Box \notin S$
  $\Rightarrow F$ is satisfiable
- The algorithm does not terminate

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF, i.e. S is a finite set of clauses

while $\Box \notin S$ and
there are clauses $C_a, C_b \in S$ and resolvent R of C_a and C_b
such that $R \notin S$ (modulo renaming)
do $S := S \cup \{R\}$

The selection of resolvents must be *fair:*
every resolvent is added eventually

Three possible behaviours:

- The algorithm terminates and $\Box \in S$
  $\Rightarrow F$ is unsatisfiable
- The algorithm terminates and $\Box \notin S$
  $\Rightarrow F$ is satisfiable
- The algorithm does not terminate
  ($\Rightarrow F$ is satisfiable)

Refinements of resolution

Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large

Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large
- Too many dead ends

Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large
- Too many dead ends
- Combinatorial explosion of the search space

Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large
- Too many dead ends
- Combinatorial explosion of the search space

Solution:

Strategies and heuristics: forbid certain resolution steps, which narrows the search space.

Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large
- Too many dead ends
- Combinatorial explosion of the search space

Solution:
Strategies and heuristics: forbid certain resolution steps, which narrows the search space.

But: Completeness must be preserved!