IN3070/4070 – Logic – Autumn 2020

Lecture 6: Unification, Normal Forms

Martin Giese

24th September 2020

Department of Informatics

University of Oslo

Today's Plan

- Unifcation
- Normal Forms
- Negation Normal Form
- Conjunctive Normal Form
- Clausal Form
- Prenex Normal Forms
- Skolemization

Outline

Unification

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to “guess” the right instantiations for x and y

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to “guess” the right instantiations for x and y
- “make both sides equal”
- Equation solving with terms!

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Let s and t be terms.

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Let s and t be terms. Find *all* substitutions that make s and t syntactically equal

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Let s and t be terms. Find *all* substitutions that make s and t syntactically equal, i.e. all σ with $\sigma(s) = \sigma(t)$.

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Let s and t be terms. Find *all* substitutions that make s and t syntactically equal, i.e. all σ with $\sigma(s) = \sigma(t)$.

- A substitution that makes s and t syntactically equal is called a unifier for s and t.

Unification

- Motivation: try proving the following

$$\forall x\, p(x, b) \implies \exists y\, p(a, y)$$

- Have to "guess" the right instantiations for x and y
- "make both sides equal"
- Equation solving with terms!

Unification problem

Let s and t be terms. Find *all* substitutions that make s and t syntactically equal, i.e. all σ with $\sigma(s) = \sigma(t)$.

- A substitution that makes s and t syntactically equal is called a unifier for s and t.
- To terms are unifiable if they have a unifier.

Examples

Are $f(x)$ and $f(a)$ unifiable?

Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:

Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:

f f

Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:

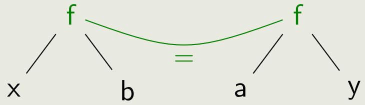
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:



- The root symbols are the same.

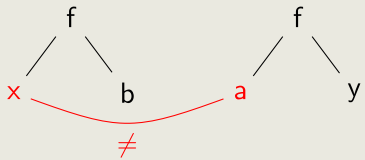
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:



- The root symbols are the same.
- The left children are different

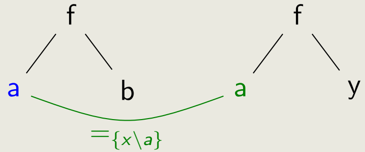
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:



- The root symbols are the same.
- The left children are different, but can be unified with $\{x \backslash a\}$.

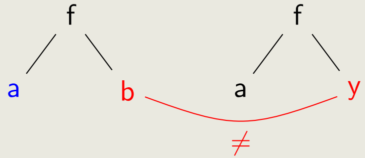
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

Easier to see if we write terms as *trees*:



- The root symbols are the same.
- The left children are different, but can be unified with $\{x \backslash a\}$.
- The right children are different

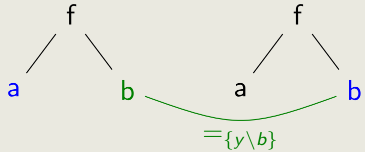
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x\backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $f(x, b)$ and $f(a, y)$ unifiable?

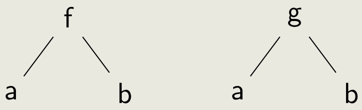
Easier to see if we write terms as *trees*:



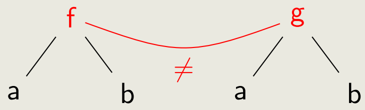
- The root symbols are the same.
- The left children are different, but can be unified with $\{x\backslash a\}$.
- The right children are different, but can be unified with $\{y\backslash b\}$.

Are $f(a, b)$ and $g(a, b)$ unifiable?

Are $f(a, b)$ and $g(a, b)$ unifiable?



Are $f(a, b)$ and $g(a, b)$ unifiable?

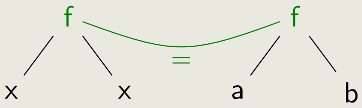


- The root symbols are different, and can *not* be unified!

Are $f(x, x)$ and $f(a, b)$ unifiable?

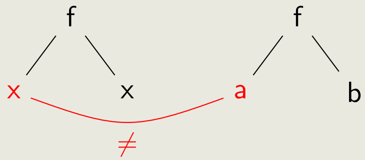
Are $f(x, x)$ and $f(a, b)$ unifiable?

Are $f(x, x)$ and $f(a, b)$ unifiable?



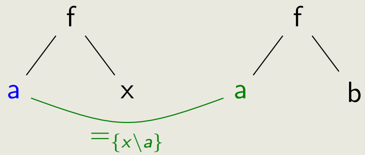
- The root symbols are equal.

Are $f(x, x)$ and $f(a, b)$ unifiable?



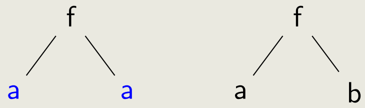
- The root symbols are equal.
- The left children are different

Are $f(x,x)$ and $f(a,b)$ unifiable?



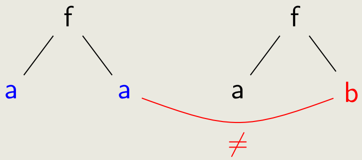
- The root symbols are equal.
- The left children are different, but can be unified with $\{x\backslash a\}$.

Are $f(x, x)$ and $f(a, b)$ unifiable?



- The root symbols are equal.
- The left children are different, but can be unified with $\{x \backslash a\}$.
- We must apply $\{x \backslash a\}$ to x in both branches.

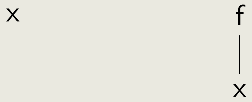
Are $f(x, x)$ and $f(a, b)$ unifiable?



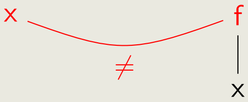
- The root symbols are equal.
- The left children are different, but can be unified with $\{x \backslash a\}$.
- We must apply $\{x \backslash a\}$ to x in both branches.
- The right children are now different, and can *not* be unified!

Are x and $f(x)$ unifiable?

Are x and $f(x)$ unifiable?

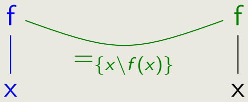


Are x and $f(x)$ unifiable?



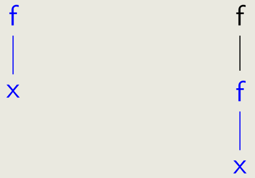
- The root symbols are different

Are x and $f(x)$ unifiable?



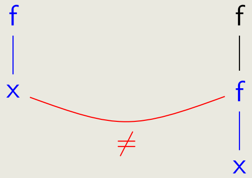
- The root symbols are different, but can be unified by $\{x \backslash f(x)\}$.

Are x and $f(x)$ unifiable?



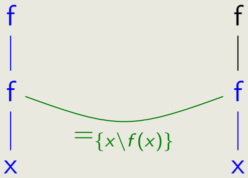
- The root symbols are different, but can be unified by $\{x \backslash f(x)\}$.
- We also have to apply $\{x \backslash f(x)\}$ on x in the right tree.

Are x and $f(x)$ unifiable?



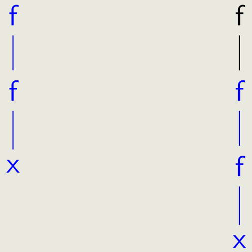
- The root symbols are different, but can be unified by $\{x \backslash f(x)\}$.
- We also have to apply $\{x \backslash f(x)\}$ on x in the right tree.
- The symbols x and f are different.

Are x and $f(x)$ unifiable?



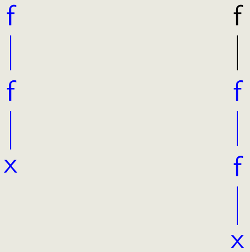
- The root symbols are different, but can be unified by $\{x\backslash f(x)\}$.
- We also have to apply $\{x\backslash f(x)\}$ on x in the right tree.
- The symbols x and f are different.
- If we unify with $\{f(x)/x\}$

Are x and $f(x)$ unifiable?



- The root symbols are different, but can be unified by $\{x\backslash f(x)\}$.
- We also have to apply $\{x\backslash f(x)\}$ on x in the right tree.
- The symbols x and f are different.
- If we unify with $\{f(x)/x\}$, we have to replace x in the right tree again.

Are x and $f(x)$ unifiable?



- The root symbols are different, but can be unified by $\{x\backslash f(x)\}$.
- We also have to apply $\{x\backslash f(x)\}$ on x in the right tree.
- The symbols x and f are different.
- If we unify with $\{f(x)/x\}$, we have to replace x in the right tree again.
- This continues indefinitely

Unification

Generally:

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

- We will define a unification algorithm, that finds all unifiers for two terms.

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

- We will define a unification algorithm, that finds all unifiers for two terms.
- Problem: Two terms can potentially have infinitely many unifiers. We can't compute all of them!

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

- We will define a unification algorithm, that finds all unifiers for two terms.
- Problem: Two terms can potentially have infinitely many unifiers. We can't compute all of them!
- Solution: Find a represetative σ for the set of unifiers, such that all other unifiers can be constructed from σ.

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

- We will define a unification algorithm, that finds all unifiers for two terms.
- Problem: Two terms can potentially have infinitely many unifiers. We can't compute all of them!
- Solution: Find a represetative σ for the set of unifiers, such that all other unifiers can be constructed from σ.
- Such a unifier is known as a most general unifier.

Composition of Substitutions

Composition of Substitutions

- Let σ and τ be substitutions.

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Definition 1.1 (Composition of Substitutions).

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Definition 1.1 (Composition of Substitutions).

Let σ and τ be substitutions.

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Definition 1.1 (Composition of Substitutions).

Let σ and τ be substitutions. The composition of σ and τ is a substitution written $\tau\sigma$

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Definition 1.1 (Composition of Substitutions).

Let σ and τ be substitutions. The composition of σ and τ is a substitution written $\tau\sigma$, such that $(\tau\sigma)(x) = \tau(\sigma(x))$ for all variables x.

Composition of Substitutions

- Let σ and τ be substitutions.
- Assume we apply first σ and then τ to a term t : $\tau(\sigma(t))$.
- The effect of this is also a substitution.

Definition 1.1 (Composition of Substitutions).

Let σ and τ be substitutions. The composition of σ and τ is a substitution written $\tau\sigma$, such that $(\tau\sigma)(x) = \tau(\sigma(x))$ for all variables x.

- Exercise: show that $(\tau\sigma)(A) = \tau(\sigma(A))$ for all formlae A and all substitutions σ and τ.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma =$$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \ldots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \ldots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \ldots, x_n\backslash\tau(s_n), \qquad\qquad \}$$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$*. Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$*.*

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{ \qquad\qquad\qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \ldots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \ldots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \ldots, x_n\backslash\tau(s_n), z_1\backslash\tau(z_1), \ldots, z_m\backslash\tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x\backslash z, y\backslash a\}$ and $\tau = \{y\backslash b, z\backslash a\}$.

Then $\tau\sigma = \{x\backslash \quad\quad\quad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z) \qquad\qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \ldots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \ldots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \ldots, x_n\backslash\tau(s_n), z_1\backslash\tau(z_1), \ldots, z_m\backslash\tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x\backslash z, y\backslash a\}$ and $\tau = \{y\backslash b, z\backslash a\}$.

Then $\tau\sigma = \{x\backslash\tau(z), y\backslash \qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a) \qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \dots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \dots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \dots, x_n\backslash\tau(s_n), z_1\backslash\tau(z_1), \dots, z_m\backslash\tau(z_m)\}$$

where $z_1, \dots, z_m$ *are the variables amongst* $y_1, \dots, y_k$ *that are not amongst* $x_1, \dots, x_n$.

Let $\sigma = \{x\backslash z, y\backslash a\}$ and $\tau = \{y\backslash b, z\backslash a\}$.

Then $\tau\sigma = \{x\backslash\tau(z), y\backslash\tau(a), z\backslash \qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \ldots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \ldots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \ldots, x_n\backslash\tau(s_n), z_1\backslash\tau(z_1), \ldots, z_m\backslash\tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x\backslash z, y\backslash a\}$ and $\tau = \{y\backslash b, z\backslash a\}$.

Then $\tau\sigma = \{x\backslash\tau(z), y\backslash\tau(a), z\backslash\tau(z)\} = \{x\backslash a, y\backslash a, z\backslash a\}$.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma = \{x \backslash \quad\quad\quad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma = \{x \backslash \tau(y) \qquad \}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma = \{x \backslash \tau(y), y \backslash \quad\}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma = \{x \backslash \tau(y), y \backslash \tau(y)\}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1\backslash s_1, \ldots, x_n\backslash s_n\}$ *and* $\tau = \{y_1\backslash t_1, \ldots, y_k\backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1\backslash\tau(s_1), \ldots, x_n\backslash\tau(s_n), z_1\backslash\tau(z_1), \ldots, z_m\backslash\tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x\backslash z, y\backslash a\}$ and $\tau = \{y\backslash b, z\backslash a\}$.

Then $\tau\sigma = \{x\backslash\tau(z), y\backslash\tau(a), z\backslash\tau(z)\} = \{x\backslash a, y\backslash a, z\backslash a\}$.

Let $\sigma = \{x\backslash y\}$ and $\tau = \{y\backslash x\}$.

Then $\tau\sigma = \{x\backslash\tau(y), y\backslash\tau(y)\} = \{x\backslash x, y\backslash x\}$

Composition of Substitutions with finite support

Proposition 1.1.

Let $\sigma = \{x_1 \backslash s_1, \ldots, x_n \backslash s_n\}$ *and* $\tau = \{y_1 \backslash t_1, \ldots, y_k \backslash t_k\}$. *Then*

$$\tau\sigma = \{x_1 \backslash \tau(s_1), \ldots, x_n \backslash \tau(s_n), z_1 \backslash \tau(z_1), \ldots, z_m \backslash \tau(z_m)\}$$

where $z_1, \ldots, z_m$ *are the variables amongst* $y_1, \ldots, y_k$ *that are not amongst* $x_1, \ldots, x_n$.

Let $\sigma = \{x \backslash z, y \backslash a\}$ and $\tau = \{y \backslash b, z \backslash a\}$.

Then $\tau\sigma = \{x \backslash \tau(z), y \backslash \tau(a), z \backslash \tau(z)\} = \{x \backslash a, y \backslash a, z \backslash a\}$.

Let $\sigma = \{x \backslash y\}$ and $\tau = \{y \backslash x\}$.

Then $\tau\sigma = \{x \backslash \tau(y), y \backslash \tau(y)\} = \{x \backslash x, y \backslash x\} = \{y \backslash x\}$.

More General Substitution

Definition 1.2 (More General Substitution).

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 *and* σ_2 *be substitutions.*

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(a), y \backslash a\}$?

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x\backslash f(y)\}$ more general than $\{x\backslash f(a), y\backslash a\}$?

Yes, since $\{x\backslash f(a), y\backslash a\} = \{y\backslash a\}\{x\backslash f(y)\}$.

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x\backslash f(y)\}$ more general than $\{x\backslash f(a), y\backslash a\}$?

Yes, since $\{x\backslash f(a), y\backslash a\} = \{y\backslash a\}\{x\backslash f(y)\}$.

Is $\{x\backslash f(a)\}$ more general than $\{x\backslash f(y)\}$?

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(a), y \backslash a\}$?

Yes, since $\{x \backslash f(a), y \backslash a\} = \{y \backslash a\}\{x \backslash f(y)\}$.

Is $\{x \backslash f(a)\}$ more general than $\{x \backslash f(y)\}$?

No, because there is no substitution τ such that $\{x \backslash f(y)\} = \tau\{x \backslash f(a)\}$.

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(a), y \backslash a\}$?

Yes, since $\{x \backslash f(a), y \backslash a\} = \{y \backslash a\}\{x \backslash f(y)\}$.

Is $\{x \backslash f(a)\}$ more general than $\{x \backslash f(y)\}$?

No, because there is no substitution τ such that $\{x \backslash f(y)\} = \tau\{x \backslash f(a)\}$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(y)\}$

More General Substitution

Definition 1.2 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x\backslash f(y)\}$ more general than $\{x\backslash f(a), y\backslash a\}$?

Yes, since $\{x\backslash f(a), y\backslash a\} = \{y\backslash a\}\{x\backslash f(y)\}$.

Is $\{x\backslash f(a)\}$ more general than $\{x\backslash f(y)\}$?

No, because there is no substitution τ such that $\{x\backslash f(y)\} = \tau\{x\backslash f(a)\}$.

Is $\{x\backslash f(y)\}$ more general than $\{x\backslash f(y)\}$

Yes, since $\{x\backslash f(y)\} = \{\}\{x\backslash f(y)\}$, where $\{\}$ is the identity substitution.

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms.

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Let $s = f(x)$ and $t = f(y)$.

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Let $s = f(x)$ and $t = f(y)$.

- $\sigma_1 = \{x \backslash a, y \backslash a\}$ is a unifier for s and t

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Let $s = f(x)$ and $t = f(y)$.

- $\sigma_1 = \{x\backslash a, y\backslash a\}$ is a unifier for s and t
- $\sigma_2 = \{x\backslash y\}$ and $\sigma_3 = \{y\backslash x\}$ are also unifiers for s and t

Most General Unifiers

Definition 1.3 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Let $s = f(x)$ and $t = f(y)$.

- $\sigma_1 = \{x \backslash a, y \backslash a\}$ is a unifier for s and t
- $\sigma_2 = \{x \backslash y\}$ and $\sigma_3 = \{y \backslash x\}$ are also unifiers for s and t
- σ_2 and σ_3 are the most general unifiers for s and t

Variable Renaming

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. *$\eta(x)$ is a variable for all $x \in \mathcal{V}$, and*

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. $\eta(x)$ *is a variable for all* $x \in \mathcal{V}$, *and*
2. $\eta(x) \neq \eta(y)$ *for all* $x, y \in \mathcal{V}$ *with* $x \neq y$.

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. $\eta(x)$ *is a variable for all* $x \in \mathcal{V}$, *and*
2. $\eta(x) \neq \eta(y)$ *for all* $x, y \in \mathcal{V}$ *with* $x \neq y$.

Are these substitutions variable renamings?

- $\sigma_1 = \{x\backslash z, y\backslash x, z\backslash y\}$
- $\sigma_2 = \{x\backslash z, z\backslash y\}$
- $\sigma_3 = \{x\backslash z, y\backslash x, z\backslash y, u\backslash a\}$

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. $\eta(x)$ *is a variable for all* $x \in \mathcal{V}$, *and*
2. $\eta(x) \neq \eta(y)$ *for all* $x, y \in \mathcal{V}$ *with* $x \neq y$.

Are these substitutions variable renamings?

- $\sigma_1 = \{x \backslash z, y \backslash x, z \backslash y\}$ Yes.
- $\sigma_2 = \{x \backslash z, z \backslash y\}$
- $\sigma_3 = \{x \backslash z, y \backslash x, z \backslash y, u \backslash a\}$

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. $\eta(x)$ *is a variable for all* $x \in \mathcal{V}$*, and*
2. $\eta(x) \neq \eta(y)$ *for all* $x, y \in \mathcal{V}$ *with* $x \neq y$.

Are these substitutions variable renamings?

- $\sigma_1 = \{x \backslash z, y \backslash x, z \backslash y\}$ Yes.
- $\sigma_2 = \{x \backslash z, z \backslash y\}$ No, because $\sigma_2(y) = \sigma_2(z)$.
- $\sigma_3 = \{x \backslash z, y \backslash x, z \backslash y, u \backslash a\}$

Variable Renaming

- The previous example shows that two terms can have several most general unifiers.
- But these mgus are always equal up to variable renaming.

Definition 1.4 (Variable Renaming).

A subsitution η is a variable renaming if

1. $\eta(x)$ *is a variable for all* $x \in \mathcal{V}$, *and*
2. $\eta(x) \neq \eta(y)$ *for all* $x, y \in \mathcal{V}$ *with* $x \neq y$.

Are these substitutions variable renamings?

- $\sigma_1 = \{x \backslash z, y \backslash x, z \backslash y\}$ Yes.
- $\sigma_2 = \{x \backslash z, z \backslash y\}$ No, because $\sigma_2(y) = \sigma_2(z)$.
- $\sigma_3 = \{x \backslash z, y \backslash x, z \backslash y, u \backslash a\}$ No, because $\sigma_3(u)$ is not a variable.

Uniqueness “up to variable renaming”

Proposition 1.2.

If σ_1 and σ_2 are most general unifiers for two terms s and t, then there is a variable renaming η such that $\eta\sigma_1 = \sigma_2$.

Uniqueness “up to variable renaming”

Proposition 1.2.

If σ_1 and σ_2 are most general unifiers for two terms s and t, then there is a variable renaming η such that $\eta\sigma_1 = \sigma_2$.

- We leave out the proof.

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms:

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms:

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

- Subterms:

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

- Subterms: x, a, $f(x, a)$

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

- Subterms: x, a, $f(x, a)$
- Strict subterms:

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

- Subterms: x, a, $f(x, a)$
- Strict subterms: x, a

Subterms

Definition 1.5.

The set of subterms of a term t is the smallest set T such that

- $t \in T$, *and*
- *if* $f(t_1, \ldots, t_n) \in T$, *then all* $t_i \in T$.

All terms in T except t are called strict subterms of t.

Let $s = gx$.

- Subterms: x, gx
- Strict subterms: x

Let $t = f(x, a)$.

- Subterms: x, a, $f(x, a)$
- Strict subterms: x, a

- So every term is a subterm of itself, but not a strict subterm.

Numbered Term Trees

Numbered Term Trees

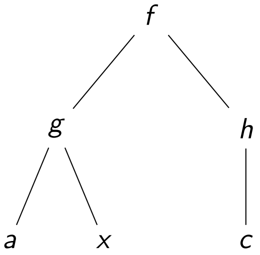
- We have seen that terms can be represented by trees.

Numbered Term Trees

- We have seen that terms can be represented by trees.
- For the unification algorithm, it is convenient to number the children of nodes:

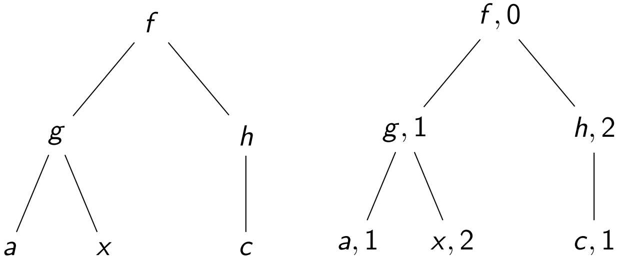
Numbered Term Trees

- We have seen that terms can be represented by trees.
- For the unification algorithm, it is convenient to number the children of nodes:



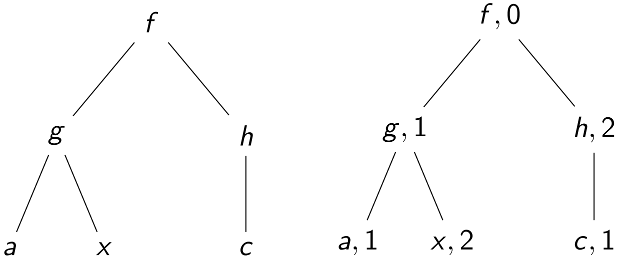
Numbered Term Trees

- We have seen that terms can be represented by trees.
- For the unification algorithm, it is convenient to number the children of nodes:



Numbered Term Trees

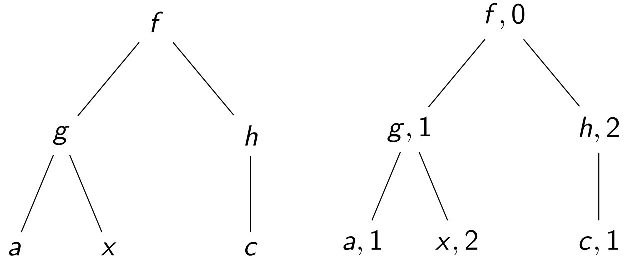
- We have seen that terms can be represented by trees.
- For the unification algorithm, it is convenient to number the children of nodes:



- We call such trees numbered term trees.

Numbered Term Trees

- We have seen that terms can be represented by trees.
- For the unification algorithm, it is convenient to number the children of nodes:



- We call such trees numbered term trees.
- We write the root of the numbered term tree of t as $\text{root}(t)$.

Critical Pair

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*
- *k_2 is a subterm of t_2*

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*
- *k_2 is a subterm of t_2*
- *when terms are considered as numbered trees,*

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*
- *k_2 is a subterm of t_2*
- *when terms are considered as numbered trees,*
  - $\text{root}(k_1)$ *is different from* $\text{root}(k_2)$

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*
- *k_2 is a subterm of t_2*
- *when terms are considered as numbered trees,*
  - $\text{root}(k_1)$ *is different from* $\text{root}(k_2)$
  - *The path from* $\text{root}(t_1)$ *to* $\text{root}(k_1)$ *is equal to the path from* $\text{root}(t_2)$ *to* $\text{root}(k_2)$

Critical Pair

- When we unify terms t_1 and t_2, we want to find subtrees that are different.
- We also want to find differeing subtrees as close to the root as possible.

Definition 1.6 (Critical Pairs).

A crtical pair for two terms t_1 and t_2 is a pair $\langle k_1, k_2 \rangle$ such that

- *k_1 is a subterm of t_1*
- *k_2 is a subterm of t_2*
- *when terms are considered as numbered trees,*
  - $\mathrm{root}(k_1)$ *is different from* $\mathrm{root}(k_2)$
  - *The path from* $\mathrm{root}(t_1)$ *to* $\mathrm{root}(k_1)$ *is equal to the path from* $\mathrm{root}(t_2)$ *to* $\mathrm{root}(k_2)$

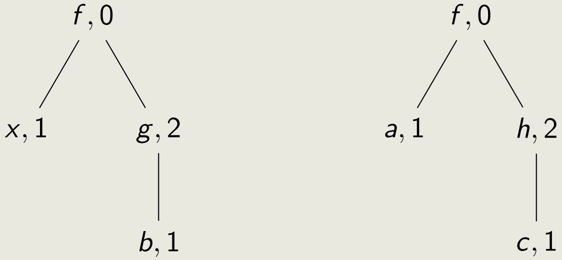
- Paths can be empty, i.e. terms differ at the root.

Example.

Let $s = f(x, gb)$ *and* $t = f(a, hc)$.

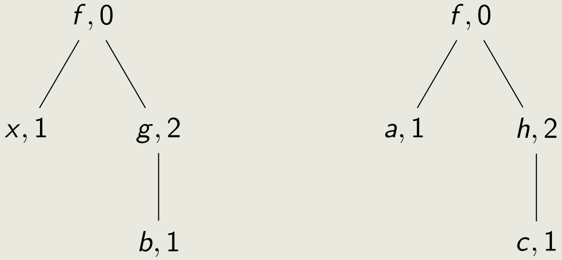
Example.

Let $s = f(x, gb)$ *and* $t = f(a, hc)$. *This gives the following numbered term trees:*



Example.

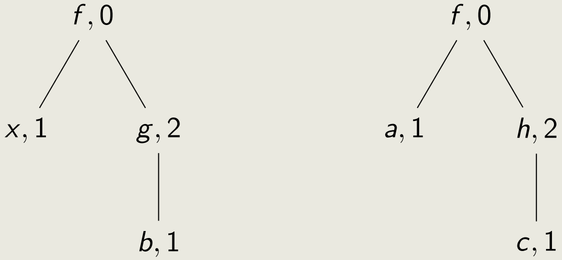
Let $s = f(x, gb)$ and $t = f(a, hc)$. This gives the following numbered term trees:



- *Is $\langle b, c \rangle$ a critical pair for s and t?*

Example.

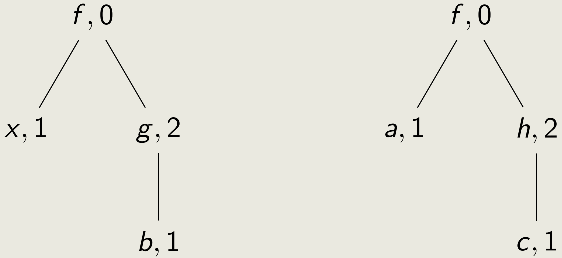
Let $s = f(x, gb)$ and $t = f(a, hc)$. This gives the following numbered term trees:



- *Is $\langle b, c \rangle$ a critical pair for s and t?*
  - *No, the path from* $\mathrm{root}(s)$ *to* $\mathrm{root}(b)$ *differs from the path from* $\mathrm{root}(t)$ *to* $\mathrm{root}(c)$*.*

Example.

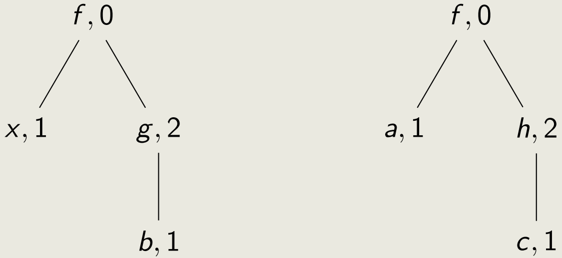
Let $s = f(x, gb)$ *and* $t = f(a, hc)$*. This gives the following numbered term trees:*



- *Is* $\langle b, c\rangle$ *a critical pair for* s *and* t*?*
  - *No, the path from* $\text{root}(s)$ *to* $\text{root}(b)$ *differs from the path from* $\text{root}(t)$ *to* $\text{root}(c)$*.*
- *Is* $\langle x, a\rangle$ *a critical pair for* s *and* t*?*

Example.

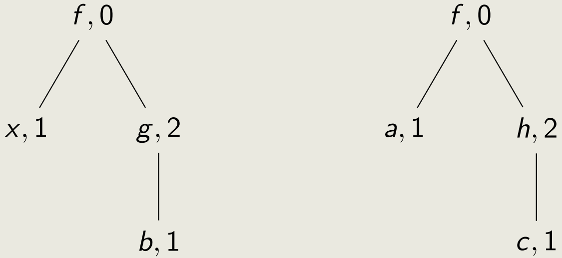
Let $s = f(x, gb)$ *and* $t = f(a, hc)$. *This gives the following numbered term trees:*



- *Is* $\langle b, c \rangle$ *a critical pair for* s *and* t*?*
  - *No, the path from* $\text{root}(s)$ *to* $\text{root}(b)$ *differs from the path from* $\text{root}(t)$ *to* $\text{root}(c)$.
- *Is* $\langle x, a \rangle$ *a critical pair for* s *and* t*? Yes.*

Example.

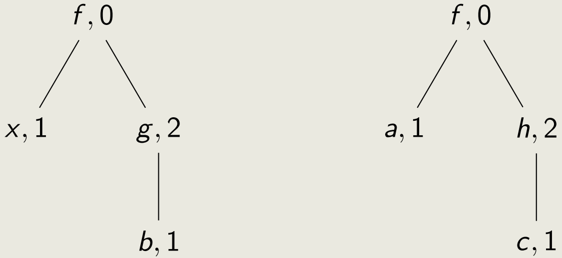
Let $s = f(x, gb)$ *and* $t = f(a, hc)$. *This gives the following numbered term trees:*



- *Is* $\langle b, c \rangle$ *a critical pair for* s *and* t?
  - *No, the path from* $\text{root}(s)$ *to* $\text{root}(b)$ *differs from the path from* $\text{root}(t)$ *to* $\text{root}(c)$.
- *Is* $\langle x, a \rangle$ *a critical pair for* s *and* t? *Yes.*
- *Is* $\langle gb, hc \rangle$ *a critical pair for* s *and* t?

Example.

Let $s = f(x, gb)$ *and* $t = f(a, hc)$. *This gives the following numbered term trees:*



- *Is* $\langle b, c\rangle$ *a critical pair for* s *and* t?
  - *No, the path from* $\text{root}(s)$ *to* $\text{root}(b)$ *differs from the path from* $\text{root}(t)$ *to* $\text{root}(c)$.
- *Is* $\langle x, a\rangle$ *a critical pair for* s *and* t? *Yes.*
- *Is* $\langle gb, hc\rangle$ *a critical pair for* s *and* t? *Yes.*

Unification Algorithm

Algoritm: unify(t_1, t_2)

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;

while $(\sigma(t_1) \neq \sigma(t_2))$ **do**

end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
    choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;

end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
    choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
    if (neither k_1 nor k_2 are variables) **then**

    end if

end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
  choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
  if (neither k_1 nor k_2 are variables) **then**
    return *"not unifiable"*;
  end if
end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
    choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
    if (neither k_1 nor k_2 are variables) **then**
        return *"not unifiable"*;
    end if
    $x :=$ the one of k_1, k_2 that is a variable (if both are, choose one)
    $t :=$ the one of k_1, k_2 that is not x;
end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
    choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
    if (neither k_1 nor k_2 are variables) **then**
        return *"not unifiable"*;
    end if
    $x :=$ the one of k_1, k_2 that is a variable (if both are, choose one)
    $t :=$ the one of k_1, k_2 that is not x;
    if (x occurs in t) **then**

    end if

end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
  choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
  if (neither k_1 nor k_2 are variables) **then**
    return *"not unifiable"*;
  end if
  x := the one of k_1, k_2 that is a variable (if both are, choose one)
  t := the one of k_1, k_2 that is not x;
  if (x occurs in t) **then**
    return *"not unifiable"*;
  end if

end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
  choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
  if (neither k_1 nor k_2 are variables) **then**
    return *"not unifiable"*;
  end if
  x := the one of k_1, k_2 that is a variable (if both are, choose one)
  t := the one of k_1, k_2 that is not x;
  if (x occurs in t) **then**
    return *"not unifiable"*;
  end if
  $\sigma := \{x \backslash t\}\sigma$;
end while

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while $(\sigma(t_1) \neq \sigma(t_2))$ **do**
  choose a critical pair $\langle k_1, k_2 \rangle$ for $\sigma(t_1), \sigma(t_2)$;
  if (neither k_1 nor k_2 are variables) **then**
    return *"not unifiable"*;
  end if
  $x :=$ the one of k_1, k_2 that is a variable (if both are, choose one)
  $t :=$ the one of k_1, k_2 that is not x;
  if (x occurs in t) **then**
    return *"not unifiable"*;
  end if
  $\sigma := \{x \backslash t\}\sigma$;
end while
return σ;

Properties of the Unification Algorithm

Properties of the Unification Algorithm

- If the terms t_1 and t_2 are unifiable, the algorithm returns a most general unifier for t_1 and t_2.

Properties of the Unification Algorithm

- If the terms t_1 and t_2 are unifiable, the algorithm returns a most general unifier for t_1 and t_2.
- The mgu is representative for all other unifiers of t_1 and t_2.

Properties of the Unification Algorithm

- If the terms t_1 and t_2 are unifiable, the algorithm returns a most general unifier for t_1 and t_2.
- The mgu is representative for all other unifiers of t_1 and t_2.
- If t_1 and t_2 are not unifiable, the algorithm returns *"not unifiable"*.

Outline

- Unifcation
- Normal Forms
- Negation Normal Form
- Conjunctive Normal Form
- Clausal Form
- Prenex Normal Forms
- Skolemization

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is ‘nice’
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is ‘nice’
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...
- Now assume that for every $a \in A$ there is a $n \in N$ with $n \approx a$.

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...
- Now assume that for every $a \in A$ there is a $n \in N$ with $n \approx a$.
- Instead of the 'ugly' a, we can work with the 'nice' n.

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...
- Now assume that for every $a \in A$ there is a $n \in N$ with $n \approx a$.
- Instead of the 'ugly' a, we can work with the 'nice' n.
- In computer science: computable function $f : A \to N$ with $f(a) \approx a$

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...
- Now assume that for every $a \in A$ there is a $n \in N$ with $n \approx a$.
- Instead of the 'ugly' a, we can work with the 'nice' n.
- In computer science: computable function $f : A \to N$ with $f(a) \approx a$
- Members of N are "in N-normal form"

What are Normal Forms?

- Given some set A of formulas, grammars, programs, etc.
- And a subset $N \subseteq A$ that is 'nice'
  - Easy to read off certain properties
  - Easy to compute with
  - Easy to write programs for
  - ...
- Given an equivalence relation $\approx$ on A
  - formulas are logically equivalent
  - grammars describe the same language
  - programs compute the same function
  - ...
- Now assume that for every $a \in A$ there is a $n \in N$ with $n \approx a$.
- Instead of the 'ugly' a, we can work with the 'nice' n.
- In computer science: computable function $f : A \to N$ with $f(a) \approx a$
- Members of N are "in N-normal form"
- For every a, we can compute the (or a) N-normal form $f(a)$.

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.
- Let $\langle m, n \rangle \approx \langle m', n' \rangle$ iff $m \cdot n' = m' \cdot n$, e.g. $\frac{3}{4} \approx \frac{6}{8}$.

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.
- Let $\langle m, n \rangle \approx \langle m', n' \rangle$ iff $m \cdot n' = m' \cdot n$, e.g. $\frac{3}{4} \approx \frac{6}{8}$.
- Reduced fractions are nice to check whether two are $\approx$: If $\langle m, n \rangle$ and $\langle m', n' \rangle$ are both reduced, then

$$\langle m, n \rangle \approx \langle m', n' \rangle \quad \Leftrightarrow \quad \langle m, n \rangle = \langle m', n' \rangle$$

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.
- Let $\langle m, n \rangle \approx \langle m', n' \rangle$ iff $m \cdot n' = m' \cdot n$, e.g. $\frac{3}{4} \approx \frac{6}{8}$.
- Reduced fractions are nice to check whether two are $\approx$: If $\langle m, n \rangle$ and $\langle m', n' \rangle$ are both reduced, then

$$\langle m, n \rangle \approx \langle m', n' \rangle \quad \Leftrightarrow \quad \langle m, n \rangle = \langle m', n' \rangle$$

- Algorithm: Given $\langle m, n \rangle$, compute $k = gcd(m, n)$, return $\langle m/k, n/k \rangle$.

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n\rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.
- Let $\langle m, n\rangle \approx \langle m', n'\rangle$ iff $m \cdot n' = m' \cdot n$, e.g. $\frac{3}{4} \approx \frac{6}{8}$.
- Reduced fractions are nice to check whether two are $\approx$: If $\langle m, n\rangle$ and $\langle m', n'\rangle$ are both reduced, then

$$\langle m, n\rangle \approx \langle m', n'\rangle \quad \Leftrightarrow \quad \langle m, n\rangle = \langle m', n'\rangle$$

- Algorithm: Given $\langle m, n\rangle$, compute $k = gcd(m, n)$, return $\langle m/k, n/k\rangle$.
- Then $\langle m/k, n/k\rangle$ is reduced and $\langle m/k, n/k\rangle \approx \langle m, n\rangle$

Example: Normal Form for Rational Numbers

- Let Q be the set of pairs $\langle m, n \rangle$, where we think of $\frac{m}{n}$
- 'nice' fractions are reduced, i.e. no common divisors in m and n
- E.g. $\frac{3}{4}$ is reduced but $\frac{6}{8}$ is not.
- Let $\langle m, n \rangle \approx \langle m', n' \rangle$ iff $m \cdot n' = m' \cdot n$, e.g. $\frac{3}{4} \approx \frac{6}{8}$.
- Reduced fractions are nice to check whether two are $\approx$: If $\langle m, n \rangle$ and $\langle m', n' \rangle$ are both reduced, then

$$\langle m, n \rangle \approx \langle m', n' \rangle \quad \Leftrightarrow \quad \langle m, n \rangle = \langle m', n' \rangle$$

- Algorithm: Given $\langle m, n \rangle$, compute $k = gcd(m, n)$, return $\langle m/k, n/k \rangle$.
- Then $\langle m/k, n/k \rangle$ is reduced and $\langle m/k, n/k \rangle \approx \langle m, n \rangle$
- So $\langle m/k, n/k \rangle$ is the "reduced normal form" of $\langle m, n \rangle$

Outline

- Unifcation
- Normal Forms
- Negation Normal Form
- Conjunctive Normal Form
- Clausal Form
- Prenex Normal Forms
- Skolemization

Negation Normal Form

Definition 3.1 (Negation Normal Form).

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

- $p \rightarrow q$ *is not in NNF*

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

- $p \rightarrow q$ *is not in NNF*
- $\neg p \vee q$ *is in NNF*

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

- $p \to q$ *is not in NNF*
- $\neg p \lor q$ *is in NNF*
- $\neg(p \lor \forall x \neg q(x))$ *is not in NNF*

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

- $p \to q$ *is not in NNF*
- $\neg p \lor q$ *is in NNF*
- $\neg(p \lor \forall x \, \neg q(x))$ *is not in NNF*
- $\neg p \land \exists x \, q(x)$ *is in NNF*

Negation Normal Form

Definition 3.1 (Negation Normal Form).

A formula is in negation normal form (NNF) if it contains no implications, and all negations are in front of literals.

Example.

- $p \rightarrow q$ *is not in NNF*
- $\neg p \vee q$ *is in NNF*
- $\neg(p \vee \forall x \neg q(x))$ *is not in NNF*
- $\neg p \wedge \exists x\, q(x)$ *is in NNF*

Theorem 3.1.

Every formula in first-order logic can be transformed into an equivalent formula in NNF.

Proof.

To convert an arbitrary formula to a formula in NNF, remove implications, and push negations inwards, preserving equivalence, using the following:

$$A \to B \equiv \neg A \lor B$$
$$\neg(A \land B) \equiv \neg A \lor \neg B$$
$$\neg(A \lor B) \equiv \neg A \land \neg B$$
$$\neg(\forall x\, A) \equiv \exists x\, \neg A$$
$$\neg(\exists x\, A) \equiv \forall x\, \neg A$$
$$\neg(\neg A) \equiv A$$

□

Advantage of Negation Normal Form

- Tableau or single-sided sequent calculi need 50% fewer rules

Advantage of Negation Normal Form

- Tableau or single-sided sequent calculi need 50% fewer rules
- No need to handle negation outside of axioms

Advantage of Negation Normal Form

- Tableau or single-sided sequent calculi need 50% fewer rules
- No need to handle negation outside of axioms
- Sound and complete calculus for propositional logic:

$$\frac{\Gamma, A, B \implies}{\Gamma, A \land B \implies}\ \land\text{-left} \qquad \frac{\Gamma, A \implies \qquad \Gamma, B \implies}{\Gamma, A \lor B \implies}\ \lor\text{-left}$$

$$\frac{}{\Gamma, A, \neg A \implies}\ \text{ax}$$

Advantage of Negation Normal Form

- Tableau or single-sided sequent calculi need 50% fewer rules
- No need to handle negation outside of axioms
- Sound and complete calculus for propositional logic:

$$\frac{\Gamma, A, B \implies}{\Gamma, A \wedge B \implies} \wedge\text{-left} \qquad \frac{\Gamma, A \implies \qquad \Gamma, B \implies}{\Gamma, A \vee B \implies} \vee\text{-left}$$

$$\frac{}{\Gamma, A, \neg A \implies} \text{ax}$$

- Soundness and completeness proofs also have fewer cases.

Outline

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

$(p \lor \neg q) \land (\neg p \lor q)$ *is in CNF.*

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

$(p \lor \neg q) \land (\neg p \lor q)$ *is in CNF.*

$(p \lor \neg q) \land (\neg p \lor (q \land q))$ *is not in CNF.*

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

$(p \vee \neg q) \wedge (\neg p \vee q)$ *is in CNF.*

$(p \vee \neg q) \wedge (\neg p \vee (q \wedge q))$ *is not in CNF.*

What about just p *or* $(p \vee q)$*?*

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

$(p \vee \neg q) \wedge (\neg p \vee q)$ *is in CNF.*

$(p \vee \neg q) \wedge (\neg p \vee (q \wedge q))$ *is not in CNF.*

What about just p *or* $(p \vee q)$*? Yes, if we consider a literal to be both a conjunction and a disjunction.*

Conjunctive Normal Form

Definition 4.1 (Conjunctive Normal Form).

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example.

$(p \lor \neg q) \land (\neg p \lor q)$ *is in CNF.*

$(p \lor \neg q) \land (\neg p \lor (q \land q))$ *is not in CNF.*

What about just p *or* $(p \lor q)$*? Yes, if we consider a literal to be both a conjunction and a disjunction.*

Theorem 4.1.

Every formula in propositional logic can be transformed into an equivalent formula in CNF.

Proof.

□

Proof.

To convert an arbitrary propositional formula to a formula in CNF perform the following steps, each of which preserves logical equivalence:

□

Proof.

To convert an arbitrary propositional formula to a formula in CNF perform the following steps, each of which preserves logical equivalence:

(1) Convert to negation normal form.

□

Proof.

To convert an arbitrary propositional formula to a formula in CNF perform the following steps, each of which preserves logical equivalence:

(1) Convert to negation normal form.

(2) Use the distributive laws to move conjunctions inside disjunctions to the outside

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

□

Outline

Clausal Form

Definition 5.1 (Clausal Form).

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal.

Clausal Form

Definition 5.1 (Clausal Form).

A clause *is a set of literals. A clause is considered to be an implicit disjunction of its literals. A* unit clause *is a clause consisting of exactly one literal. The empty set of literals is the* empty clause*, denoted by* $\square$.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal. The empty set of literals is the empty clause, denoted by $\Box$. A formula in clausal form is a set of clauses.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal. The empty set of literals is the empty clause, denoted by $\square$. A formula in clausal form is a set of clauses. A formula is considered to be an implicit conjunction of its clauses.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal. The empty set of literals is the empty clause, denoted by $\square$. A formula in clausal form is a set of clauses. A formula is considered to be an implicit conjunction of its clauses. The formula that is the empty set of clauses is denoted by $\emptyset$.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal. The empty set of literals is the empty clause, denoted by $\square$. A formula in clausal form is a set of clauses. A formula is considered to be an implicit conjunction of its clauses. The formula that is the empty set of clauses is denoted by $\emptyset$.

The only significant difference between clausal form and the standard syntax is that clausal form is defined in terms of sets.

Clausal Form

Definition 5.1 (Clausal Form).

A clause is a set of literals. A clause is considered to be an implicit disjunction of its literals. A unit clause is a clause consisting of exactly one literal. The empty set of literals is the empty clause, denoted by $\square$. A formula in clausal form is a set of clauses. A formula is considered to be an implicit conjunction of its clauses. The formula that is the empty set of clauses is denoted by $\emptyset$.

The only significant difference between clausal form and the standard syntax is that clausal form is defined in terms of sets.

$(p \vee \neg q) \wedge (\neg p \vee q)$ in clausal form: $\{\{p, \neg q\}, \{\neg p, q\}\}$

Transformation to Clausal Form

Corollary 5.1.

Every formula ϕ *in propositional logic can be transformed into an logically equivalent formula in clausal form.*

Transformation to Clausal Form

Corollary 5.1.

Every formula ϕ in propositional logic can be transformed into an logically equivalent formula in clausal form.

Proof.

□

Transformation to Clausal Form

Corollary 5.1.

Every formula ϕ in propositional logic can be transformed into an logically equivalent formula in clausal form.

Proof.

This follows from the previous theorem, where we transformed a formula to CNF.

□

Transformation to Clausal Form

Corollary 5.1.

Every formula ϕ in propositional logic can be transformed into an logically equivalent formula in clausal form.

Proof.

This follows from the previous theorem, where we transformed a formula to CNF. Each disjunction is then transformed to a clause (of literals), and the clausal form is the set of these clauses. □

Empty Clause and Empty Set of Clauses

Lemma 5.1.

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\square$, *the empty clause, is unsatisfiable.*

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\square$, the empty clause, is unsatisfiable.

$\emptyset$, the empty set of clauses, is valid.

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\square$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

$\square$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\Box$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true.

$\Box$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\square$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true. Let $\mathcal{I}$ be an arbitrary interpretation.

$\square$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\Box$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true. Let $\mathcal{I}$ be an arbitrary interpretation. Since there are no literals in $\Box$, there are no literals whose value is true under $\mathcal{I}$.

$\Box$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\square$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true. Let $\mathcal{I}$ be an arbitrary interpretation. Since there are no literals in $\square$, there are no literals whose value is true under $\mathcal{I}$. But $\mathcal{I}$ was an arbitrary interpretation, so $\square$ is unsatisfiable.

$\square$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\Box$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true. Let $\mathcal{I}$ be an arbitrary interpretation. Since there are no literals in $\Box$, there are no literals whose value is true under $\mathcal{I}$. But $\mathcal{I}$ was an arbitrary interpretation, so $\Box$ is unsatisfiable.

A set of clauses is valid iff every clause in the set is true in every interpretation.

$\Box$

Empty Clause and Empty Set of Clauses

Lemma 5.1.

$\Box$, *the empty clause, is unsatisfiable.*

$\emptyset$, *the empty set of clauses, is valid.*

Proof.

A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true. Let $\mathcal{I}$ be an arbitrary interpretation. Since there are no literals in $\Box$, there are no literals whose value is true under $\mathcal{I}$. But $\mathcal{I}$ was an arbitrary interpretation, so $\Box$ is unsatisfiable.

A set of clauses is valid iff every clause in the set is true in every interpretation. But there are no clauses in $\emptyset$ that need be true, so $\emptyset$ is valid. $\Box$

Short Hand Notation for Clauses

Notation

Short Hand Notation for Clauses

Notation

- $\{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ means $(p \lor r) \land (\neg q \lor \neg p \lor q) \land (p \lor \neg p \lor q)$.

Short Hand Notation for Clauses

Notation

- $\{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ means $(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q)$.
- S usually denotes a formula in clausal form.

Short Hand Notation for Clauses

Notation

- $\{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ means $(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q)$.
- S usually denotes a formula in clausal form.
- C usually denotes a clause.

Short Hand Notation for Clauses

Notation

- $\{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ means $(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q)$.
- S usually denotes a formula in clausal form.
- C usually denotes a clause.
- l usually denotes a literal.

Short Hand Notation for Clauses

Notation

- $\{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ means $(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q)$.
- S usually denotes a formula in clausal form.
- C usually denotes a clause.
- l usually denotes a literal.
- l^c then represents its *complement*.

Outline

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall / \exists$) and M is a quantifier-free formula in CNF.

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall/\exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall/\exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Example.

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall/\exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Example.

$$\forall x \forall y ((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$$

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall / \exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Example.

$\forall x \forall y ((p(x,y) \lor \neg p(y,x)) \land (q(x,y) \lor \neg q(y,x)))$ *is in PCNF.*

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall / \exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Example.

$\forall x \forall y((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$ *is in PCNF.*

$\forall x(\neg p(x) \vee \exists y q(y))$

Prenex Conjunctive Normal Form

Definition 6.1 (Prenex Conjunctive Normal Form).

A formula is in prenex conjunctive normal form (PCNF) iff it is of the form:

$$Q_1 x_1 \cdots Q_n x_n M$$

where the Q_i are quantifiers ($\forall / \exists$) and M is a quantifier-free formula in CNF. The sequence $Q_1 x_1 \cdots Q_n x_n$ is the prefix and M is the matrix.

Example.

$\forall x \forall y ((p(x, y) \vee \neg p(y, x)) \wedge (q(x, y) \vee \neg q(y, x)))$ *is in PCNF.*

$\forall x (\neg p(x) \vee \exists y q(y))$ *is not in PCNF.*

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers.

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Example.

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Example.

$$\forall x \forall y ((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$$

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Example.

$$\forall x \forall y ((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$$

can be written in clausal form as

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Example.

$$\forall x \forall y((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$$

can be written in clausal form as

$$\{\{p(x,y), \neg p(y,x)\}, \{q(x,y), \neg q(y,x)\}\}$$

Clausal Form for First-order Formulae

Definition 6.2 (Clausal Form).

Let A be a closed formula in PCNF whose prefix consists only of universal quantifiers. The clausal form of A consists of the matrix of A written as a set of clauses.

Example.

$$\forall x \forall y ((p(x,y) \vee \neg p(y,x)) \wedge (q(x,y) \vee \neg q(y,x)))$$

can be written in clausal form as

$$\{\{p(x,y), \neg p(y,x)\}, \{q(x,y), \neg q(y,x)\}\}$$

Note: *The universal quantifiers are implicit.*

Outline

Skolem's Theorem

Theorem 7.1 (Skolem).

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

The notation $A \approx A'$ means that A is satisfiable if and only if A' is satisfiable.

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

The notation $A \approx A'$ means that A is satisfiable if and only if A' is satisfiable. This is not the same as logical equivalence.

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

The notation $A \approx A'$ means that A is satisfiable if and only if A' is satisfiable. This is not the same as logical equivalence. We call it equisatisfiability.

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

The notation $A \approx A'$ means that A is satisfiable if and only if A' is satisfiable. This is not the same as logical equivalence. We call it equisatisfiability.

Named after the Norwegian mathematician and logician Thoralf Albert Skolem (1887–1963).

Skolem's Theorem

Theorem 7.1 (Skolem).

There is an algorithm that for any closed formula A computes a formula A' in clausal form such that $A \approx A'$.

The notation $A \approx A'$ means that A is satisfiable if and only if A' is satisfiable. This is not the same as logical equivalence. We call it equisatisfiability.

Named after the Norwegian mathematician and logician Thoralf Albert Skolem (1887–1963).

"Satisfiability is more interesting than validity. Always true or always false are extremes."

Skolem's Algorithm

Algorithm for obtaining A':

Skolem's Algorithm

Algorithm for obtaining A':

- Rename bound variables so that no variable appears in two quantifiers.

Skolem's Algorithm

Algorithm for obtaining A':

- Rename bound variables so that no variable appears in two quantifiers.
- Transform to negation normal form

Skolem's Algorithm

Algorithm for obtaining A':

- Rename bound variables so that no variable appears in two quantifiers.
- Transform to negation normal form
- Extract quantifiers from the matrix until all quantifiers appear in the prefix and the matrix is quantifier-free.

$$
\begin{aligned}
A \wedge \forall x\, B &\equiv \forall x(A \wedge B) && \text{if } x \text{ not free in } A \\
A \wedge \exists x\, B &\equiv \exists x(A \wedge B) && \text{if } x \text{ not free in } A \\
A \vee \forall x\, B &\equiv \forall x(A \vee B) && \text{if } x \text{ not free in } A \\
A \vee \exists x\, B &\equiv \exists x(A \vee B) && \text{if } x \text{ not free in } A
\end{aligned}
$$

Skolem's Algorithm

Algorithm for obtaining A':

- Rename bound variables so that no variable appears in two quantifiers.
- Transform to negation normal form
- Extract quantifiers from the matrix until all quantifiers appear in the prefix and the matrix is quantifier-free.

$$
\begin{aligned}
A \wedge \forall x\, B &\equiv \forall x(A \wedge B) && \text{if } x \text{ not free in } A \\
A \wedge \exists x\, B &\equiv \exists x(A \wedge B) && \text{if } x \text{ not free in } A \\
A \vee \forall x\, B &\equiv \forall x(A \vee B) && \text{if } x \text{ not free in } A \\
A \vee \exists x\, B &\equiv \exists x(A \vee B) && \text{if } x \text{ not free in } A
\end{aligned}
$$

- Use the distributive laws to transform the matrix into CNF.

Skolem's Algorithm

Algorithm for obtaining A':

- Rename bound variables so that no variable appears in two quantifiers.
- Transform to negation normal form
- Extract quantifiers from the matrix until all quantifiers appear in the prefix and the matrix is quantifier-free.

$$A \land \forall x\, B \equiv \forall x (A \land B) \qquad \text{if } x \text{ not free in } A$$
$$A \land \exists x\, B \equiv \exists x (A \land B) \qquad \text{if } x \text{ not free in } A$$
$$A \lor \forall x\, B \equiv \forall x (A \lor B) \qquad \text{if } x \text{ not free in } A$$
$$A \lor \exists x\, B \equiv \exists x (A \lor B) \qquad \text{if } x \text{ not free in } A$$

- Use the distributive laws to transform the matrix into CNF.
- The formula is now in PCNF.

Skolem's Algorithm (cont.)

Algorithm for obtaining A' (continued):

Skolem's Algorithm (cont.)

Algorithm for obtaining A' (continued):

- For every existential quantifier $\exists x$ in the prefix, let $y_1, \ldots, y_n$ be the universally quantified variables preceding $\exists x$ and let f be a new n-ary function symbol.

Skolem's Algorithm (cont.)

Algorithm for obtaining A' (continued):

- For every existential quantifier $\exists x$ in the prefix, let $y_1, \ldots, y_n$ be the universally quantified variables preceding $\exists x$ and let f be a new n-ary function symbol.
- Delete $\exists x$ and replace every occurrence of x by $f(y_1, \ldots, y_n)$.

Skolem's Algorithm (cont.)

Algorithm for obtaining A' (continued):

- For every existential quantifier $\exists x$ in the prefix, let $y_1, \ldots, y_n$ be the universally quantified variables preceding $\exists x$ and let f be a new n-ary function symbol.
- Delete $\exists x$ and replace every occurrence of x by $f(y_1, \ldots, y_n)$.
- If there are no universal quantifiers preceding $\exists x$, replace x by a new constant (0-ary function).

Skolem's Algorithm (cont.)

Algorithm for obtaining A' (continued):

- For every existential quantifier $\exists x$ in the prefix, let $y_1, \ldots, y_n$ be the universally quantified variables preceding $\exists x$ and let f be a new n-ary function symbol.
- Delete $\exists x$ and replace every occurrence of x by $f(y_1, \ldots, y_n)$.
- If there are no universal quantifiers preceding $\exists x$, replace x by a new constant (0-ary function).
- These new function symbols are Skolem functions and the process of replacing existential quantifiers by functions is Skolemization.

Skolemization Example

Example.

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.
- *Are they equivalent?*

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.
- *Are they equivalent? No!*

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.
- *Are they equivalent? No!*
- *Are they equisatisfiable?*

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.
- *Are they equivalent? No!*
- *Are they equisatisfiable? Yes!*

Skolemization Example

Example.

- *Look at the formulas* $\forall x \exists y p(x, y)$ *and* $\forall x p(x, f(x))$.
- *Are they equivalent? No!*
- *Are they equisatisfiable? Yes!*
- *The Skolemization of* $\forall x \exists y p(x, y)$ *is* $\forall x p(x, f(x))$, *and if one of them has a model, so does the other.*

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.
- We must show that there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \forall y_1 \cdots \forall y_n\, A[x \backslash f(y_1, \ldots, y_n)]$).

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.
- We must show that there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \forall y_1 \cdots \forall y_n\, A[x \backslash f(y_1, \ldots, y_n)]$).
- Let $\mathcal{I}' = (D, \iota')$ such that ι' extends ι with the interpretation of f.

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.
- We must show that there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \forall y_1 \cdots \forall y_n\, A[x \backslash f(y_1, \ldots, y_n)])$.
- Let $\mathcal{I}' = (D, \iota')$ such that ι' extends ι with the interpretation of f.
- Remember that f does not occur in A, so f^{ι} does not matter

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.
- We must show that there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \forall y_1 \cdots \forall y_n\, A[x \backslash f(y_1, \ldots, y_n)])$.
- Let $\mathcal{I}' = (D, \iota')$ such that ι' extends ι with the interpretation of f.
- Remember that f does not occur in A, so f^ι does not matter
- For any choice of elements $d_1, \ldots, d_n$ from D, there is an element d_{n+1} in D such that

$$v_{\mathcal{I}}(\alpha\{y_1 \leftarrow d_1\} \cdots \{y_n \leftarrow d_n\}\{x \leftarrow d_{n+1}\}, A) = T$$

Proof of Skolem's Theorem

- The first transformations of the algorithm (into PCNF) preserve equivalence.
- We need to consider the replacement of an existential quantifier by a Skolem function.
- Suppose that $\mathcal{I} \models \forall y_1 \cdots \forall y_n \exists x\, A$ for $\mathcal{I} = (D, \iota)$.
- We must show that there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \forall y_1 \cdots \forall y_n\, A[x \backslash f(y_1, \ldots, y_n)])$.
- Let $\mathcal{I}' = (D, \iota')$ such that ι' extends ι with the interpretation of f.
- Remember that f does not occur in A, so f^ι does not matter
- For any choice of elements $d_1, \ldots, d_n$ from D, there is an element d_{n+1} in D such that

$$v_{\mathcal{I}}(\alpha\{y_1 \leftarrow d_1\} \cdots \{y_n \leftarrow d_n\}\{x \leftarrow d_{n+1}\}, A) = T$$

- Let $f^{\iota'}(d_1, \ldots, d_n) = d_{n+1}$. This ensures that the claim holds.

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$

Example

- Clause form of $\neg \exists x \, (p(x) \rightarrow \forall y \, p(y))$
- First, transform to (equivalent) Prenex Normal Form

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$$

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$\begin{aligned} & \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\ \equiv\; & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \end{aligned}$$

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
&\neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ &\forall x\,\neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ &\forall x\,(p(x) \land \neg\forall y\, p(y))
\end{aligned}
$$

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
& \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\; & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\; & \forall x\,(p(x) \land \neg\forall y\, p(y)) \\
\equiv\; & \forall x\,(p(x) \land \exists y\, \neg p(y))
\end{aligned}
$$

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
& \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\; & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\; & \forall x\, (p(x) \land \neg\forall y\, p(y)) \\
\equiv\; & \forall x\, (p(x) \land \exists y\, \neg p(y)) \\
\equiv\; & \forall x\, \exists y\, (p(x) \land \neg p(y))
\end{aligned}
$$

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
& \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\, (p(x) \land \neg\forall y\, p(y)) \\
\equiv\ & \forall x\, (p(x) \land \exists y\, \neg p(y)) \\
\equiv\ & \forall x\, \exists y\, (p(x) \land \neg p(y))
\end{aligned}
$$

- Then skolemise (preserving satisfiability)

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
& \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\,(p(x) \land \neg\forall y\, p(y)) \\
\equiv\ & \forall x\,(p(x) \land \exists y\, \neg p(y)) \\
\equiv\ & \forall x\, \exists y\,(p(x) \land \neg p(y))
\end{aligned}
$$

- Then skolemise (preserving satisfiability)

$$\forall x\,(p(x) \land \neg p(f(x)))$$

Example

- Clause form of $\neg\exists x\,(p(x) \to \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
&\neg\exists x\,(p(x) \to \forall y\, p(y)) \\
\equiv\ &\forall x\, \neg(p(x) \to \forall y\, p(y)) \\
\equiv\ &\forall x\,(p(x) \land \neg\forall y\, p(y)) \\
\equiv\ &\forall x\,(p(x) \land \exists y\, \neg p(y)) \\
\equiv\ &\forall x\, \exists y\,(p(x) \land \neg p(y))
\end{aligned}
$$

- Then skolemise (preserving satisfiability)

$$\forall x\,(p(x) \land \neg p(f(x)))$$

- In clause form, two clauses:

Example

- Clause form of $\neg\exists x\,(p(x) \rightarrow \forall y\, p(y))$
- First, transform to (equivalent) Prenex Normal Form

$$
\begin{aligned}
& \neg\exists x\,(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\, \neg(p(x) \rightarrow \forall y\, p(y)) \\
\equiv\ & \forall x\,(p(x) \wedge \neg\forall y\, p(y)) \\
\equiv\ & \forall x\,(p(x) \wedge \exists y\, \neg p(y)) \\
\equiv\ & \forall x\, \exists y\,(p(x) \wedge \neg p(y))
\end{aligned}
$$

- Then skolemise (preserving satisfiability)

$$\forall x\,(p(x) \wedge \neg p(f(x)))$$

- In clause form, two clauses:

$$\{\{p(x)\}, \{\neg p(f(x)\}\}$$

Outlook

- We have seen the LK calculus for propositional and first-order logic
- Sound and complete, but not machine-oriented
- Machine-oriented calculi use:
  - Unification to find the right instantiations
  - Normal forms to simplify reasoning steps
- Free variable calculi
  - Similar to LK, but with unification
  - Often used with NNF or clause form
  - Not this year
- Resolution
  - Basis of many theorem provers, uses unification
  - Almost always on clause form