DTU Course 02156 Logical Systems and Logic Programming (2021)

Week	Date	Main Topics (Prolog Programming in All Lessons)
35 #01	31/8	Course Prerequisites & Tutorial on Logical Systems and Logic Programming
36 #02	7/9	Chapter 1 - Introduction (Prolog Note)
37 #03	14/9	Chapter 2 - Propositional Logic: Formulas, Models, Tableaux
38 #04	21/9	Chapter 3 - Propositional Logic: Deductive Systems
39 #05	28/9	"Isabelle" - Propositional Logic: Sequent Calculus Verifier (SeCaV)
40 #06	5/10	Chapter 4 - Propositional Logic: Resolution
41 #07	12/10	Chapter 7 - First-Order Logic: Formulas, Models, Tableaux
42		(Autumn Vacation)
43 #08	26/10	Chapter 8 - First-Order Logic: Deductive Systems
44 #09	2/11	"Isabelle" - First-Order Logic: Sequent Calculus Verifier (SeCaV)
45 #10	9/11	Chapter 9 - First-Order Logic: Terms and Normal Forms
46 #11	16/11	Chapter 10 - First-Order Logic: Resolution
47 #12	23/11	Chapter 11 - First-Order Logic: Logic Programming
48 #13	30/11	Chapter 12 - First-Order Logic: Undecidability and Model Theory & Course Evaluation

Responsible: Associate Professor Jørgen Villadsen <jovi@dtu.dk>

Assignments & Exam MUST BE SOLVED INDIVIDUALLY

Assignment-1 Deadline Sunday 26/9 (Available Wednesday 15/9)

Assignment-2 Deadline Sunday 10/10 (Available Wednesday 29/9)

Assignment-3 Deadline Sunday 31/10 (Available Wednesday 13/10)

Assignment-4 Deadline Sunday 14/11 (Available Wednesday 3/11)

Assignment-5 Deadline Thursday 2/12 (Available Wednesday 17/11)

Written Exam Tuesday 14/12 (2 Hours / No Computer / All Notes Allowed)

The mandatory assignments and the written exam are evaluated as a whole – even if you do well in the mandatory assignments then you still must do decent in the written exam in order to pass the course!

A TEACHER MUST IMMEDIATELY REPORT ANY SUSPICION OF CHEATING TO THE STUDY ADMINISTRATION FOR FURTHER ACTIONS

Agenda — Week #10

Prolog — More Findall

Prolog note — The Clause Database

First-Order Logic (FOL) — Again

Skolemization — Required for Resolution

Findall Example I

```
?- member(X:Y,[b:1,a:4,a:2]).

X = b
Y = 1 ;

X = a
Y = 4 ;

X = a
Y = 2 ;

No
```

Findall Example II

```
?- findall(Y,member(X:Y,[b:1,a:4,a:2]),S).

S = [1, 4, 2] ;

No
```

```
?- bagof(Y,member(X:Y,[b:1,a:4,a:2]),S).

X = a
S = [4, 2] ;

X = b
S = [1] ;

No
```

Findall Example III

```
?- findall(X:S,bagof(Y,member(X:Y,[b:1,a:4,a:2]),S),T).

T = [a:[4, 2], b:[1]] ;

No
```

Remember that `findall(?Template,+Goal,?Bag)` creates a list of the instantiations `Template` gets successively on backtracking over `Goal` and unifies the result with `Bag`

And `findall` succeeds with an empty list if `Goal` has no solutions!

Findall Example III

```
?- findall(X:S,bagof(Y,member(X:Y,[b:1,a:4,a:2]),S),T).

T = [a:[4, 2], b:[1]] ;

No
```

Remember that `findall(?Template,+Goal,?Bag)` creates a list of the instantiations `Template` gets successively on backtracking over `Goal` and unifies the result with `Bag`

And `findall` succeeds with an empty list if `Goal` has no solutions!

On `bagof`: If `Goal` has free variables not shared with `Template` then `bagof` backtracks over the alternatives of these free variables and unifies the corresponding instantiations of `Template` with `Bag`

Note that `bagof` fails if `Goal` has no solutions!

Findall Example III

```
?- findall(X:S,bagof(Y,member(X:Y,[b:1,a:4,a:2]),S),T).

T = [a:[4, 2], b:[1]] ;

No
```

Remember that `findall(?Template,+Goal,?Bag)` creates a list of the instantiations `Template` gets successively on backtracking over `Goal` and unifies the result with `Bag`

And `findall` succeeds with an empty list if `Goal` has no solutions!

On `bagof`: If `Goal` has free variables not shared with `Template` then `bagof` backtracks over the alternatives of these free variables and unifies the corresponding instantiations of `Template` with `Bag`

Note that `bagof` fails if `Goal` has no solutions!

Finally `setof` is the same as `bagof` but sorts without duplicates

Another Findall Example I

A Prolog program is said to be deterministic if and only if it does not succeed more than once

It must not succeed more than once for any query

`member` and `append` are not deterministic

`write` and `nl` are deterministic

Another Findall Example II

```
?- deterministic((member(X,[]), write(X), nl)).

Yes

?- deterministic((member(X,[a]), write(X), nl)).
a

Yes

?- deterministic((member(X,[a,b]), write(X), nl)).
a
b

No
```

Another Findall Example III

```
?- deterministic((member(X,[a,b,c]), write(X), nl)).
a
b
c

No
```

```
deterministic(G) :- findall(_,G,L), length(L,N), N =< 1.
```

```
?- deterministic((member(X,_), write(X), nl)).
...
```

Another Findall Example III

```
?- deterministic((member(X,[a,b,c]), write(X), nl)).
a
b
c

No
```

```
deterministic(G) :- findall(_,G,L), length(L,N), N =< 1.
```

```
?- deterministic((member(X,_), write(X), nl)).
...
```

Not the best program...

More on Variables

Special predicates to test for “free” variables and equality:

More on Variables

Special predicates to test for "free" variables and equality:

`var(?Term)` succeeds iff `Term` currently is uninstantiated.

More on Variables

Special predicates to test for “free” variables and equality:

`var(?Term)` succeeds iff `Term` currently is uninstantiated.

`nonvar(?Term)` succeeds iff `Term` currently is instantiated.

More on Variables

Special predicates to test for “free” variables and equality:

`var(?Term)` succeeds iff `Term` currently is uninstantiated.

`nonvar(?Term)` succeeds iff `Term` currently is instantiated.

`?Term1 == ?Term2` succeeds iff `Term1` and `Term2` are equal.

More on Variables

Special predicates to test for “free” variables and equality:

`var(?Term)` succeeds iff `Term` currently is uninstantiated.

`nonvar(?Term)` succeeds iff `Term` currently is instantiated.

`?Term1 == ?Term2` succeeds iff `Term1` and `Term2` are equal.

`?Term1 \== ?Term2` succeeds iff `Term1` and `Term2` are unequal.

```
?- var(X).

Yes
```

```
?- var(f(X)).

No
```

```
?- X == X.

Yes
```

```
?- X == Y.

No
```

Even More on Variables

The standard order of terms is used for equality — recall that the order of variables is system dependent — but unified variables are equal.

```
?- X == X.

Yes
```

```
?- X == Y.

No
```

```
?- _ == _.

No
```

```
?- X == Y, X = Y.

No
```

```
?- X = Y, X == Y.

X = Y

Yes
```

```
?- _ = _.

Yes
```

The Clause Database

Normally ground facts are added to the clause database:

```
?- asserta(p(a)).

Yes

?- p(X).

X = a ;

No
```

`asserta(+Term)` adds `Term` to the database as the first fact or clause of the corresponding predicate.

`assertz(+Term)` adds `Term` to the database as the last fact or clause of the corresponding predicate.

The Clause Database — Assert & Retract

In SWI-Prolog `assert` is the same as `assertz` (but `assert` is not in ISO Prolog).

It is also possible to remove the added facts (and clauses, but here only facts — in particular ground facts — will be considered):

```
?- asserta(p(a)), assertz(p(b)), retract(p(X)).

X = a ;

X = b ;

No
```

`retract(+Term)` is unified with the first unifying fact or clause in the database and the fact or clause is removed from the database.

A Simple Counter — Example

```
?- start.

Yes

?- counter(X), count, counter(Y).

X = 0
Y = 1

Yes

?- count, count, count, counter(Z).

Z = 4

Yes
```

A Simple Counter — Implementation

```
start :-
  asserta(counter(0)).

count :-
  retract(counter(N)),
  N1 is N+1,
  asserta(counter(N1)).

stop :-
  retract(counter(_)).
```

In SWI-Prolog `retractall` can be used instead of `retract` to retract all clauses instead of only the first one (but `retractall` is not in ISO Prolog).

Generation of Symbols — Example

SWI-Prolog has a special auto-loaded predicate:

```
?- gensym(f,X).

X = f1 ;

No

?- gensym(f,X).

X = f2 ;

No
```

Generation of Symbols — Implementation

It can be programmed using the clause database:

```
gensym(F,A) :-
  ( retract(counter(N)) -> N1 is N+1 ; N1 = 1 ),
  asserta(counter(N1)), atom_concat(F,N1,A).
```

There are several special predicates for manipulation of atoms.

```
?- atom_concat(abc,def,X).

X = abcdef ;

No
```

It is similar to `append` although for atoms rather than lists.

Generation of Symbols — Atom Concatenation

```
?- atom_concat(X,Y,abc).

X = ''
Y = abc ;

X = a
Y = bc ;

X = ab
Y = c ;

X = abc
Y = '' ;

No
```

Retractall

The deterministic `retractall/1` built-in predicate in SWI-Prolog behaves as if it were defined as follows:

```
retractall(Term) :- retract(Term), fail.
retractall(Term) :- retract((Term :- _)), fail.
retractall(_).
```

Hence it retracts (removes) all facts or clauses in the clause database for which the head unifies with the argument of the `retractall/1` predicate.

Syntax — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

Syntax — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

$term ::= x \mid a \mid f(term\_list)$

Syntax — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

$term ::= x \mid a \mid f(term\_list)$

$term\_list ::= term \mid term, term\_list$

Syntax — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

$term ::= x \mid a \mid f(term\_list)$

$term\_list ::= term \mid term, term\_list$

$atomic\_formula ::= p \mid p(term\_list)$

Syntax — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

$\mathit{term} ::= x \mid a \mid f(\mathit{term\_list})$

$\mathit{term\_list} ::= \mathit{term} \mid \mathit{term}, \mathit{term\_list}$

$\mathit{atomic\_formula} ::= p \mid p(\mathit{term\_list})$

$$\begin{aligned}\mathit{formula} ::= \;& \mathit{atomic\_formula} \mid \neg \mathit{formula} \mid \\ & \mathit{formula} \wedge \mathit{formula} \mid \mathit{formula} \vee \mathit{formula} \mid \\ & \ldots \mid \\ & \forall x \mathit{formula} \mid \exists x \mathit{formula}\end{aligned}$$

Interpretations & Assignments — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

Interpretations & Assignments — Functions in FOL

Example: $\forall x \forall y\, p(f(a,y), g(x, h(x), y))$

Let U be a set of formulas such that $\{p_1, \ldots, p_k\}$ are all the predicates, $\{f_1, \ldots, f_l\}$ are all the functions and $\{a_1, \ldots, a_m\}$ are all the constants appearing in U.

Interpretations & Assignments — Functions in FOL

Example: $\forall x \forall y\, p(f(a,y), g(x, h(x), y))$

Let U be a set of formulas such that $\{p_1, \ldots, p_k\}$ are all the predicates, $\{f_1, \ldots, f_l\}$ are all the functions and $\{a_1, \ldots, a_m\}$ are all the constants appearing in U.

An *interpretation* $\mathcal{I}$ is a 4-tuple

$$(D, \{R_1, \ldots, R_k\}, \{F_1, \ldots, F_l\}, \{d_1, \ldots, d_m\})$$

where D is a *non-empty* domain, R_i is an assignment of an n_i-ary relation on D to the n_i-ary predicate p_i, F_i is an assignment of an n_i-ary function on D to the n_i-ary function f_i, and d_i is an assignment of an element of D to the constant a_i.

Interpretations & Assignments — Functions in FOL

Example: $\forall x \forall y\, p(f(a,y), g(x,h(x),y))$

Let U be a set of formulas such that $\{p_1,\ldots,p_k\}$ are all the predicates, $\{f_1,\ldots,f_l\}$ are all the functions and $\{a_1,\ldots,a_m\}$ are all the constants appearing in U.

An *interpretation* $\mathcal{I}$ is a 4-tuple

$$(D, \{R_1,\ldots,R_k\}, \{F_1,\ldots,F_l\}, \{d_1,\ldots,d_m\})$$

where D is a *non-empty* domain, R_i is an assignment of an n_i-ary relation on D to the n_i-ary predicate p_i, F_i is an assignment of an n_i-ary function on D to the n_i-ary function f_i, and d_i is an assignment of an element of D to the constant a_i.

Given an interpretation $\mathcal{I}$ an *assignment* $\sigma_{\mathcal{I}}$ is a function which maps every variable to an element of the domain of $\mathcal{I}$.

Example: $\forall x \forall y \, p(f(a, y), g(x, h(x), y))$

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y \, p(f(a, y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

If A is closed then $v_{\sigma_{\mathcal{I}}}(A)$ does not depend on $\sigma_{\mathcal{I}}$ but only on $\mathcal{I}$.

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y\, p(f(a,y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

If A is closed then $v_{\sigma_{\mathcal{I}}}(A)$ does not depend on $\sigma_{\mathcal{I}}$ but only on $\mathcal{I}$.

Hence for a closed formula A the value is denoted $v_{\mathcal{I}}(A)$.

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y \, p(f(a, y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

If A is closed then $v_{\sigma_{\mathcal{I}}}(A)$ does not depend on $\sigma_{\mathcal{I}}$ but only on $\mathcal{I}$.

Hence for a closed formula A the value is denoted $v_{\mathcal{I}}(A)$.

A closed formula A is *valid*, denoted $\models A$, iff $v_{\mathcal{I}}(A) = T$ for *all* interpretations $\mathcal{I}$.

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y\, p(f(a, y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

If A is closed then $v_{\sigma_{\mathcal{I}}}(A)$ does not depend on $\sigma_{\mathcal{I}}$ but only on $\mathcal{I}$.

Hence for a closed formula A the value is denoted $v_{\mathcal{I}}(A)$.

A closed formula A is *valid*, denoted $\vDash A$, iff $v_{\mathcal{I}}(A) = T$ for *all* interpretations $\mathcal{I}$.

Several proof systems $\vdash A$ are available:

- Tableaux
- Gentzen and Hilbert systems
- Resolution

Semantics & Proof Systems — Functions in FOL

Example: $\forall x \forall y\, p(f(a,y), g(x, h(x), y))$

Let A be a formula and let $v_{\sigma_{\mathcal{I}}}(A)$ be the value of A under $\sigma_{\mathcal{I}}$.

If A is closed then $v_{\sigma_{\mathcal{I}}}(A)$ does not depend on $\sigma_{\mathcal{I}}$ but only on $\mathcal{I}$.

Hence for a closed formula A the value is denoted $v_{\mathcal{I}}(A)$.

A closed formula A is *valid*, denoted $\models A$, iff $v_{\mathcal{I}}(A) = T$ for *all* interpretations $\mathcal{I}$.

Several proof systems $\vdash A$ are available:

- Tableaux
- Gentzen and Hilbert systems
- Resolution

Soundness and Completeness Theorem: $\models A$ **iff** $\vdash A$

Valuation — Functions in FOL

$v_{\sigma_{\mathcal{I}}}(A)$ is defined as follows:

$v_{\sigma_{\mathcal{I}}}(p_i(t_1, \ldots\ t_n)) = T$ iff $(v_{\sigma_{\mathcal{I}}}(t_1), \ldots, v_{\sigma_{\mathcal{I}}}(t_n)) \in R_i$

$v_{\sigma_{\mathcal{I}}}(\neg A_1) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = F$

$v_{\sigma_{\mathcal{I}}}(A_1 \wedge A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ and $v_{\sigma_{\mathcal{I}}}(A_2) = T$

$v_{\sigma_{\mathcal{I}}}(A_1 \vee A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ or $v_{\sigma_{\mathcal{I}}}(A_2) = T$

...

$v_{\sigma_{\mathcal{I}}}(\forall x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *all* $d \in D$

$v_{\sigma_{\mathcal{I}}}(\exists x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *some* $d \in D$

Valuation — Functions in FOL

$v_{\sigma_{\mathcal{I}}}(A)$ is defined as follows:

$v_{\sigma_{\mathcal{I}}}(p_i(t_1, \ldots\ t_n)) = T$ iff $(v_{\sigma_{\mathcal{I}}}(t_1), \ldots, v_{\sigma_{\mathcal{I}}}(t_n)) \in R_i$

$v_{\sigma_{\mathcal{I}}}(\neg A_1) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = F$

$v_{\sigma_{\mathcal{I}}}(A_1 \wedge A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ and $v_{\sigma_{\mathcal{I}}}(A_2) = T$

$v_{\sigma_{\mathcal{I}}}(A_1 \vee A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ or $v_{\sigma_{\mathcal{I}}}(A_2) = T$

...

$v_{\sigma_{\mathcal{I}}}(\forall x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *all* $d \in D$

$v_{\sigma_{\mathcal{I}}}(\exists x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *some* $d \in D$

$v_{\sigma_{\mathcal{I}}}(x) = d$ iff $\sigma_{\mathcal{I}}$ maps x to d

Valuation — Functions in FOL

$v_{\sigma_\mathcal{I}}(A)$ is defined as follows:

$v_{\sigma_\mathcal{I}}(p_i(t_1, \ldots\ t_n)) = T$ iff $(v_{\sigma_\mathcal{I}}(t_1), \ldots, v_{\sigma_\mathcal{I}}(t_n)) \in R_i$

$v_{\sigma_\mathcal{I}}(\neg A_1) = T$ iff $v_{\sigma_\mathcal{I}}(A_1) = F$

$v_{\sigma_\mathcal{I}}(A_1 \wedge A_2) = T$ iff $v_{\sigma_\mathcal{I}}(A_1) = T$ and $v_{\sigma_\mathcal{I}}(A_2) = T$

$v_{\sigma_\mathcal{I}}(A_1 \vee A_2) = T$ iff $v_{\sigma_\mathcal{I}}(A_1) = T$ or $v_{\sigma_\mathcal{I}}(A_2) = T$

...

$v_{\sigma_\mathcal{I}}(\forall x A_1) = T$ iff $v_{\sigma_\mathcal{I}[x \leftarrow d]}(A_1) = T$ for *all* $d \in D$

$v_{\sigma_\mathcal{I}}(\exists x A_1) = T$ iff $v_{\sigma_\mathcal{I}[x \leftarrow d]}(A_1) = T$ for *some* $d \in D$

$v_{\sigma_\mathcal{I}}(x) = d$ iff $\sigma_\mathcal{I}$ maps x to d

$v_{\sigma_\mathcal{I}}(a_i) = d_i$

Valuation — Functions in FOL

$v_{\sigma_{\mathcal{I}}}(A)$ is defined as follows:

$v_{\sigma_{\mathcal{I}}}(p_i(t_1, \ldots\ t_n)) = T$ iff $(v_{\sigma_{\mathcal{I}}}(t_1), \ldots, v_{\sigma_{\mathcal{I}}}(t_n)) \in R_i$

$v_{\sigma_{\mathcal{I}}}(\neg A_1) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = F$

$v_{\sigma_{\mathcal{I}}}(A_1 \wedge A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ and $v_{\sigma_{\mathcal{I}}}(A_2) = T$

$v_{\sigma_{\mathcal{I}}}(A_1 \vee A_2) = T$ iff $v_{\sigma_{\mathcal{I}}}(A_1) = T$ or $v_{\sigma_{\mathcal{I}}}(A_2) = T$

...

$v_{\sigma_{\mathcal{I}}}(\forall x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *all* $d \in D$

$v_{\sigma_{\mathcal{I}}}(\exists x A_1) = T$ iff $v_{\sigma_{\mathcal{I}}[x \leftarrow d]}(A_1) = T$ for *some* $d \in D$

$v_{\sigma_{\mathcal{I}}}(x) = d$ iff $\sigma_{\mathcal{I}}$ maps x to d

$v_{\sigma_{\mathcal{I}}}(a_i) = d_i$

$v_{\sigma_{\mathcal{I}}}(f_i(t_1, \ldots, t_n)) = F_i(v_{\sigma_{\mathcal{I}}}(t_1), \ldots, v_{\sigma_{\mathcal{I}}}(t_n))$

Teaser — Resolution for First-Order Logic

Is $p(a) \rightarrow \exists x p(x)$ valid?

$\boxed{\models p(a) \rightarrow \exists x p(x)}$

Teaser — Resolution for First-Order Logic

Is $p(a) \to \exists x p(x)$ valid? $\boxed{\models p(a) \to \exists x p(x)}$

Negated formula	$\neg(p(a) \to \exists x p(x))$
Rename bound variables	(no change)
Eliminate boolean operators	$\neg(\neg p(a) \lor \exists x p(x))$
Push negation inwards	$p(a) \land \forall x \neg p(x)$
Extract quantifiers	$\forall x (p(a) \land \neg p(x))$
Distribute matrix	(no change)
Replace existential quantifiers	(no change)

$S_0 = \{\{p(a)\}, \{\neg p(x)\}\}$.

Teaser — Resolution for First-Order Logic

Is $p(a) \to \exists x p(x)$ valid? $\boxed{\models p(a) \to \exists x p(x)}$

Negated formula	$\neg(p(a) \to \exists x p(x))$
Rename bound variables	(no change)
Eliminate boolean operators	$\neg(\neg p(a) \vee \exists x p(x))$
Push negation inwards	$p(a) \wedge \forall x \neg p(x)$
Extract quantifiers	$\forall x(p(a) \wedge \neg p(x))$
Distribute matrix	(no change)
Replace existential quantifiers	(no change)

$S_0 = \{\{p(a)\}, \{\neg p(x)\}\}$.

$\square$ is obtained since $p(a)$ and $p(x)$ have most general unifier $x = a$.

Teaser — Resolution for First-Order Logic

Is $p(a) \to \exists x p(x)$ valid? $\boxed{\models p(a) \to \exists x p(x)}$

Negated formula	$\neg(p(a) \to \exists x p(x))$
Rename bound variables	(no change)
Eliminate boolean operators	$\neg(\neg p(a) \vee \exists x p(x))$
Push negation inwards	$p(a) \wedge \forall x \neg p(x)$
Extract quantifiers	$\forall x (p(a) \wedge \neg p(x))$
Distribute matrix	(no change)
Replace existential quantifiers	(no change)

$S_0 = \{\{p(a)\}, \{\neg p(x)\}\}$.

$\Box$ is obtained since $p(a)$ and $p(x)$ have most general unifier $x = a$.

Since the empty clause is produced for the negated formula, the original formula is valid.

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

For example: $(\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p)$

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

For example: $(\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p)$

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 (the prefix) and M is a quantifier-free formula in CNF (the matrix).

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

For example: $(\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p)$

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 (the prefix) and M is a quantifier-free formula in CNF (the matrix).

A closed formula is in *clausal form* iff it is in PCNF with only universal quantifiers.

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

For example: $(\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p)$

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 (the prefix) and M is a quantifier-free formula in CNF (the matrix).

A closed formula is in *clausal form* iff it is in PCNF with only universal quantifiers.

Theorem (Skolem): For any closed formula A there exists a formula A' in clausal form such that A is satisfiable iff A' is satisfiable.

Clausal Form

Recall that a formula is in *conjunctive normal form* (CNF) iff it consists of conjunctions of disjunctions of literals.

For example: $(\neg q \lor \neg p \lor q) \land (p \lor \neg p \lor q \lor p \lor \neg p)$

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 (the prefix) and M is a quantifier-free formula in CNF (the matrix).

A closed formula is in *clausal form* iff it is in PCNF with only universal quantifiers.

Theorem (Skolem): For any closed formula A there exists a formula A' in clausal form such that A is satisfiable iff A' is satisfiable.

Note that it need not be the case that $A \equiv A'$.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

There is an algorithm that given A produces some A' as described in the theorem.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

There is an algorithm that given A produces some A' as described in the theorem.

New functions — Skolem functions — can be introduced by the algorithm corresponding to existentially quantified variables.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

There is an algorithm that given A produces some A' as described in the theorem.

New functions — Skolem functions — can be introduced by the algorithm corresponding to existentially quantified variables.

The Skolem functions replaces all occurrences of the existentially quantified variable.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

There is an algorithm that given A produces some A' as described in the theorem.

New functions — Skolem functions — can be introduced by the algorithm corresponding to existentially quantified variables.

The Skolem functions replaces all occurrences of the existentially quantified variable.

The arity of a Skolem function corresponds to the number of universally quantified variables *preceding* (in the PCNF transformation) the existentially quantified variable.

Skolemization — Idea

Thoralf Albert Skolem (1887 - 1963) was a Norwegian mathematician / logician.

Skolemization is the first step in resolution for FOL.

There is an algorithm that given A produces some A' as described in the theorem.

New functions — Skolem functions — can be introduced by the algorithm corresponding to existentially quantified variables.

The Skolem functions replaces all occurrences of the existentially quantified variable.

The arity of a Skolem function corresponds to the number of universally quantified variables *preceding* (in the PCNF transformation) the existentially quantified variable.

No universally quantified variables *preceding* the existentially quantified variable introduces a new constant (0-ary function).

Skolemization — Example

Rename, Eliminate, De Morgan, Extract, Distribute, <u>Skolemization</u>

Skolemization — Example

Rename, Eliminate, De Morgan, Extract, Distribute, Skolemization

```
?- skolem(all(X, p(X) => q(X)) =>
          (all(X, p(X)) => all(X, q(X)))).
(Ax1(p(x1) => q(x1)) => (Ax1p(x1) => Ax1q(x1)))
(Ax1(p(x1) => q(x1)) => (Ax2p(x2) => Ax3q(x3)))
(~Ax1(~p(x1) \ q(x1)) \ (~Ax2p(x2) \ Ax3q(x3)))
(Ex1(p(x1) & ~q(x1)) \ (Ex2~p(x2) \ Ax3q(x3)))
Ex1Ex2Ax3((p(x1) & ~q(x1)) \ (~p(x2) \ q(x3)))
Ex1Ex2Ax3((p(x1) \ (~p(x2) \ q(x3))) &
          (~q(x1) \ (~p(x2) \ q(x3))))
Ax1((p(f1) \ (~p(f2) \ q(x1))) &
    (~q(f1) \ (~p(f2) \ q(x1))))
[p(f1),~p(f2),q(x1)][~q(f1),~p(f2),q(x1)]

Yes
```

Skolemization — Another Example

```
?- skolem(ex(X, all(Y, p(X,Y))) =>
          all(Y, ex(X, p(X,Y)))).
(Ex1Ax2p(x1,x2) => Ax2Ex1p(x1,x2))
(Ex1Ax2p(x1,x2) => Ax3Ex4p(x4,x3))
(~Ex1Ax2p(x1,x2) \ Ax3Ex4p(x4,x3))
(Ax1Ex2~p(x1,x2) \ Ax3Ex4p(x4,x3))
Ax1Ex2Ax3Ex4(~p(x1,x2) \ p(x4,x3))
Ax1Ex2Ax3Ex4(~p(x1,x2) \ p(x4,x3))
Ax1Ax2(~p(x1,f1(x1)) \ p(f2(x2,x1),x2))
[~p(x1,f1(x1)),p(f2(x2,x1),x2)]

Yes
```

Sometimes “shorter” alternative transformation are possible:

`[~p(x1,f1(x1)),p(f2(x2),x2)]`

Push quantifiers inwards before the Skolem functions replacement