Knowledge Representation and Reasoning
Logics for Artificial Intelligence

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu

Contents

Part I

Part II

Part III

4 Full First-Order Predicate Logic (FOL)

4.1 CarPool World

We'll add Tom and Betty's mothers:

$motherOf(Tom)$ and $motherOf(Betty)$

CarPool World Domain Rules (Partial)

$$\forall x(Driver(x) \Rightarrow \neg Passenger(x))$$

$$\forall x, y(Drives(x, y) \Rightarrow (Driver(x) \wedge Passenger(y)))$$

4.2 The "Standard" First-Order Predicate Logic

4.2.1 Syntax of the "Standard" First-Order Predicate Logic Atomic Symbols

Individual Constants:

- Any letter of the alphabet (preferably early),
- any (such) letter with a numeric subscript,
- any character string not containing blanks nor other punctuation marks.

For example: a, B_{12}, Tom, *Tom's_mother-in-law*.

Atomic Symbols, Part 2

Arbitrary Individuals:

- Any letter of the alphabet (preferably early),
- any (such) letter with a numeric subscript.

Indefinite Individuals:

- Any letter of the alphabet (preferably early),
- any (such) letter with a numeric subscript.

Atomic Symbols, Part 3

Variables:

- Any letter of the alphabet (preferably late),
- any (such) letter with a numeric subscript.

For example: x, y_6.

Atomic Symbols, Part 4

Function Symbols:

- Any letter of the alphabet (preferably early middle)
- any (such) letter with a numeric subscript
- any character string not containing blanks.

For example: $f, g_2, motherOf, familyOf$.

Atomic Symbols, Part 5

Predicate Symbols:

- Any letter of the alphabet (preferably late middle),
- any (such) letter with a numeric subscript,
- any character string not containing blanks.

For example: P, Q_4, $Passenger$, $Drives$.

Atomic Symbols, Part 6

Each Function Symbol and Predicate Symbol must have a particular **arity**.

Use superscript for explicit arity.

For example: $motherOf^{1}, Drives^{2}, familyOf^{2}, g_2^3$

Atomic Symbols, Part 7

In any specific predicate logic language

Individual Constants,
Arbitrary Individuals,
Indefinite Individuals,
Variables,
Function Symbols,
Predicate Symbols

must be disjoint.

Terms

- Every individual constant, every arbitrary individual, every indefinite individual, and every variable is a term.
- If f^n is a function symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $f^n(t_1, \ldots, t_n)$ is a term.
  (The superscript may be omitted if no confusion results.)
  For example: $\mathit{familyOf}^2(\mathit{Tom}, \mathit{motherOf}^1(\mathit{Betty}))$
- Nothing else is a term.

Atomic Formulas

If P^n is a predicate symbol of arity n,

and $t_1, \ldots, t_n$ are terms,

then $P^n(t_1, \ldots, t_n)$ is an atomic formula.
E.g.: $\mathit{ChildIn}^2(\mathit{Betty}, \mathit{familyOf}^2(\mathit{Tom}, \mathit{motherOf}^1(\mathit{Betty})))$

(The superscript may be omitted if no confusion results.)

Well-Formed Formulas (wffs):

- Every atomic formula is a wff.
- If P is a wff, then so is $\neg(P)$.
- If P and Q are wffs, then so are

  $(P \wedge Q)$ $\quad$ $(P \vee Q)$

  $(P \Rightarrow Q)$ $\quad$ $(P \Leftrightarrow Q)$
- If P is a wff and x is a variable, then $\forall x(P)$ and $\exists x(P)$ are wffs.

  Parentheses may be omitted or replaced by square brackets if no confusion results.

  We will allow $(P_1 \wedge \cdots \wedge P_n)$ and $(P_1 \vee \cdots \vee P_n)$.

  $\forall x(\forall y(P))$ may be abbreviated as $\forall x, y(P)$.
  $\exists x(\exists y(P))$ may be abbreviated as $\exists x, y(P)$.

Open, Closed, Ground, and Free For

A wff with a free variable is called **open**.

A wff with no free variables is called **closed**,

An expression with no variables is called **ground**.

Note: expressions now include functional terms.

A term t is **free for** a variable x in the wff $A(x)$ if
no free occurrence of x in $A(x)$ is in the scope
of any quantifier $\forall y$ or $\exists y$ whose variable y is in t.
E.g., $f(a, y, b)$ is free for x in $\forall u \exists v(A(x, u) \vee B(x, v))$
but $f(a, y, b)$ is not free for x in $\forall u \exists y(A(x, u) \vee B(x, y))$.
Remedy: rename y in $A(x)$. E.g., $\forall u \exists v(A(x, u) \vee B(x, v))$

Substitutions with Functional Terms

Notice, terms may now include functional terms.

E.g.:

$$P(x, f(y), z)\{a/x, g(b)/y, f(a)/z\} = P(a, f(g(b)), f(a))$$

4.2.2 Semantics of the "Standard" First-Order Predicate Logic

Assumes a **Domain**, $\mathcal{D}$, of

- individuals,
- functions on individuals,
- sets of individuals,
- relations on individuals

Let $\mathcal{I}$ be set of all individuals in $\mathcal{D}$.

Semantics of Constants

Individual Constant:

$[a] = [\![a]\!]$ = some particular individual in $\mathcal{I}$.

Arbitrary Individual:

$[a] = [\![a]\!]$ = a representative of all individuals in $\mathcal{I}$. Everything True about all of them, is True of it.

Indefinite Individual:

$[s] = [\![s]\!]$ = a representative of some individual in $\mathcal{I}$, but it's unspecified which one.

There is no anonymous individual.

I.e. for every individual, i in $\mathcal{I}$, there is a ground term t such that $[\![t]\!] = \mathrm{i}$. (But not necessarily an individual constant.)

Intensional Semantics of Functional Terms

Function Symbols: $[f^n]$ is some n-ary function in $\mathcal{D}$,

Functional Terms:

If f^n is some function symbol and $t_1, \ldots, t_n$ are ground terms, then $[f^n(t_1, \ldots, t_n)]$ is a description of the individual in $\mathcal{I}$ that is the value of $[f^n]$ on $[t_1]$, and $\ldots$, and $[t_n]$.

Extensional Semantics of Functional Terms

Function Symbols: $[\![f^n]\!]$ is some function in $\mathcal{D}$,

$[\![f^n]\!]$: $\underbrace{\mathcal{I} \times \cdots \times \mathcal{I}}_{n \text{ times}} \to \mathcal{I}$

Functional Terms:

If f^n is some function symbol and $t_1, \ldots, t_n$ are ground terms, then $[\![f^n(t_1, \ldots, t_n)]\!] = [\![f^n]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.

Semantics of Predicate Symbols

Predicate Symbols:

- $[P^1]$ is some category/property of individuals of $\mathcal{I}$
- $[P^n]$ is some n-ary relation in $\mathcal{D}$.
- $[\![P^1]\!]$ is some particular subset of $\mathcal{I}$.
- $[\![P^n]\!]$ is some particular subset of the relation

$$\underbrace{\mathcal{I} \times \cdots \times \mathcal{I}}_{n \text{ times}}.$$

Intensional Semantics of Ground Atomic Formulas

- If P^1 is some unary predicate symbol,
  and t is some ground term,
  then $[P^1(t)]$ is the proposition that $[t]$ is an instance of the category $[P^1]$ (or has the property $[P^1]$).
- If P^n is some n-ary predicate symbol,
  and $t_1, \ldots, t_n$ are ground terms,
  then $[P^n(t_1, \ldots, t_n)]$ is the proposition that the relation $[P^n]$ holds among individuals $[t_1]$, and $\ldots$, and $[t_n]$.

Extensional Semantics of Ground Atomic Formulas

Atomic Formulas:

- If P^1 is some unary predicate symbol,
  and t is some ground term,
  then $[\![P^1(t)]\!]$ is True if $[\![t]\!] \in [\![P^1]\!]$,
  and False otherwise.
- If P^n is some n-ary predicate symbol,
  and $t_1, \ldots, t_n$ are ground terms,
  then $[\![P^n(t_1, \ldots, t_n)]\!]$ is True
  if $\langle [\![t_1]\!], \ldots, [\![t_n]\!] \rangle \in [\![P^n]\!]$,
  and False otherwise.

Semantics of WFFs, Part 1

$[\neg P]$, $[P \wedge Q]$, $[P \vee Q]$, $[P \Rightarrow Q]$, $[P \Leftrightarrow Q]$
$[\![\neg P]\!]$, $[\![P \wedge Q]\!]$, $[\![P \vee Q]\!]$, $[\![P \Rightarrow Q]\!]$, and $[\![P \Leftrightarrow Q]\!]$
are as they are in Propositional Logic.

Semantics of WFFs, Part 2

- $[\forall x P]$ is the proposition that every individual i in $\mathcal{I}$, with name or description t_i, satisfies $[P\{t_i/x\}]$.
- $[\exists x P]$ is the proposition that some individual i in $\mathcal{I}$, with name or description t_i, satisfies $[P\{t_i/x\}]$.
- $[\![\forall x P]\!]$ is True if $[\![P\{t/x\}]\!]$ is True for every ground term, t. Otherwise, it is False.
- $[\![\exists x P]\!]$ is True if there is some ground term, t such that $[\![P\{t/x\}]\!]$ is True. Otherwise, it is False.

Intensional Semantics of a 2-Car CarPool World 1

Individual Constants:

$[Tom]$ = The individual named Tom.

$[Betty]$ = The individual named Betty.

Functions:

$[motherOf(x)]$ = The mother of $[x]$.

Intensional Semantics of a 2-Car CarPool World 2

Predicates:

$[Driver^1(x)] = [x]$ is the driver of a car.

$[Passenger^1(x)] = [x]$ is the passenger in a car.

$[Drives^2(x, y)] = [x]$ drives $[y]$ in a car.

Extensional Semantics of a 2-Car CarPool World Situation

$[\![Tom]\!]$ = the individual named Tom.

$[\![Betty]\!]$ = the individual named Betty.

$[\![motherOf]\!] = \{\langle\ [\![Betty]\!], [\![motherOf(Betty)]\!]\ \rangle,$
$\langle\ [\![Tom]\!], [\![motherOf(Tom)]\!]\rangle\}.$

$[\![Driver]\!] = \{[\![motherOf(Betty)]\!], [\![motherOf(Tom)]\!]\}.$

$[\![Passenger]\!] = \{[\![Betty]\!], [\![Tom]\!]\}.$

$[\![Drives]\!] = \{\langle\ [\![motherOf(Betty)]\!], [\![Betty]\!]\ \rangle,$
$\langle\ [\![motherOf(Tom)]\!], [\![Tom]\!]\ \rangle\}.$

4.2.3 Model Checking in Full FOL

n Individual Constants.

At least one function yields ∞ terms.$^{*Decreasoner.}$
E.g., $motherOf(Tom), motherOf(motherOf(Tom)),$
$motherOf(motherOf(motherOf(Tom))) \ldots$.

So ∞ ground atomic propositions.

So ∞ situations (columns of truth table).

So can't create entire truth table.

Can't do model checking
by expanding quantified expressions
into Boolean combination of ground wffs.

There still could be a finite domain if at least one individual in $\mathcal{I}$ has an ∞ number of terms describing it, but we'll assume not.

4.2.4 Hilbert-Style Proof Theory for First-Order Predicate Logic

(A1). $(\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A}))$

(A2). $((\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})) \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C})))$

(A3). $((\neg\mathcal{B} \Rightarrow \neg\mathcal{A}) \Rightarrow ((\neg\mathcal{B} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{B}))$

(A4). $\forall x\mathcal{A} \Rightarrow \mathcal{A}\{t/x\}$

where t is any term free for x in $A(x)$.

(A5). $(\forall x(\mathcal{A} \Rightarrow \mathcal{B})) \Rightarrow (\mathcal{A} \Rightarrow \forall x\mathcal{B})$

if $\mathcal{A}$ is a wff containing no free occurrences of x.

Hilbert-Style Rules of Inference for "Standard" First-Order Predicate Logic

$$\frac{\mathcal{A}, \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}}$$

$$\frac{\mathcal{A}}{\forall x \mathcal{A}}$$

Note: $\exists x \mathcal{A}$ is just an abbreviation of $\neg \forall x \neg \mathcal{A}$.

4.2.5 Fitch-Style Proof Theory for First-Order Predicate Logic

Additional Rules of Inference for $\forall$

$$\begin{array}{rll|l} i & & a & \text{Arb I} \\ & & \vdots & \\ j & & P(a) & \\ \hline j+1 & \forall x P\{x/a\} & \forall I, i\text{–}j & \end{array}$$

$$\begin{array}{rl|ll} i & & \forall x P(x) & \\ i+1 & P\{t/x\} & \forall E, i & \end{array}$$

Where a is an arbitrary individual not otherwise used in the proof, and t is any term, whether or not used elsewhere in the proof, that is free for x in $P(x)$.

Example of $\forall$ Rules

To prove $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$

1	$\forall x(P(x) \Rightarrow Q(x))$	Hyp
2	$\forall x P(x)$	Hyp
3	a	Arb I
4	$\forall x P(x)$	Reit, 2
5	$P(a)$	$\forall$E, 4
6	$\forall x(P(x) \Rightarrow Q(x))$	Reit, 1
7	$P(a) \Rightarrow Q(a)$	$\forall$E, 6
8	$Q(a)$	$\Rightarrow$E, 5,7
9	$\forall x Q(x)$	$\forall$I, 3–8
10	$\forall x P(x) \Rightarrow \forall x Q(x)$	$\Rightarrow$I, 2–9
11	$\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$	$\Rightarrow$I, 1–10

Additional Rules of Inference for $\exists$

$$\begin{array}{rl} i & P(t) \\ i+1 & \exists x P(x) \quad \exists I, i \end{array}$$

$$\begin{array}{rll} i & \exists x P(x) & \\ & \vdots & \\ j & \quad P\{a/x\} & \text{Indef I}, i \\ & \quad \vdots & \\ k & \quad Q & \\ k+1 & Q & \exists E, j\text{–}k \end{array}$$

Where $P(x)$ is the result of replacing some or all occurrences of t in $P(t)$ by x, t is free for x in $P(x)$;

a is an indefinite individual not otherwise used in the proof,

$P(a/x)$ is the result of replacing all occurrences of x in $P(x)$ by a,

and there is no occurrence of a in Q. (Compare $\exists E$ to $\vee E$.)

Example of $\exists$ Rules

To prove $\exists x(P(x) \wedge Q(x)) \Rightarrow (\exists x P(x) \wedge \exists x Q(x))$

1	$\exists x(P(x) \wedge Q(x))$	Hyp
2	$\quad P(a) \wedge Q(a)$	Indef I, 1
3	$\quad P(a)$	$\wedge$E, 2
4	$\quad \exists x P(x)$	$\exists$I, 3
5	$\exists x P(x)$	$\exists$E, 2–4
6	$\quad P(b) \wedge Q(b)$	Indef I, 1
7	$\quad Q(b)$	$\wedge$E, 5
8	$\quad \exists x Q(x)$	$\exists$I, 6
9	$\exists x Q(x)$	$\exists$E, 5–7
10	$\exists x P(x) \wedge \exists x Q(x)$	$\wedge$I, 5,9
11	$\exists x(P(x) \wedge Q(x)) \Rightarrow (\exists x P(x) \wedge \exists x Q(x))$	$\Rightarrow$I, 1–10

CarPool Situation Derivation

1	$\forall x(Driver(x) \Rightarrow \neg Passenger(x))$	
2	$\forall x \forall y(Drives(x, y) \Rightarrow (Driver(x) \wedge Passenger(y)))$	
3	$\forall x Drives(motherOf(x), x)$	Hyp
4	$Drives(motherOf(Tom), Tom)$	$\forall E, 3$
5	$\forall y(Drives(motherOf(Tom), y)$	
	$\Rightarrow (Driver(motherOf(Tom)) \wedge Passenger(y)))$	$\forall E, 2$
6	$Drives(motherOf(Tom), Tom)$	
	$\Rightarrow (Driver(motherOf(Tom)) \wedge Passenger(Tom))$	$\forall E, 5$
7	$Driver(motherOf(Tom)) \wedge Passenger(Tom)$	$\Rightarrow E, 4, 6$
8	$Driver(motherOf(Tom))$	$\wedge E, 7$
9	$\exists x Driver(motherOf(x))$	$\exists I, 8$

4.3 Clause-Form First-Order Predicate Logic

4.3.1 Syntax of Clause-Form First-Order Predicate Logic
Atomic Symbols

Individual Constants:

- Any letter of the alphabet (preferably early),
- any (such) letter with a numeric subscript,
- any character string not containing blanks nor other punctuation marks.

For example: a, B_{12}, Tom, *Tom's_mother-in-law*.

Skolem Constants: Look like individual constants.

Atomic Symbols, Part 2

Variables:

- Any letter of the alphabet (preferably late),
- any (such) letter with a numeric subscript.

For example: u, v_6.

Atomic Symbols, Part 3

Function Symbols:

- Any letter of the alphabet (preferably early middle)
- any (such) letter with a numeric subscript
- any character string not containing blanks.

For example: f, g_2.
Use superscript for explicit arity.

Skolem Function Symbols: Look like function symbols.

Atomic Symbols, Part 4

Predicate Symbols:

- Any letter of the alphabet (preferably late middle),
- any (such) letter with a numeric subscript,
- any character string not containing blanks.

For example: P, Q_4, odd.

Use superscript for explicit arity.

Terms

- Every individual constant, every Skolem constant, and every variable is a term.
- If f^n is a function symbol or Skolem function symbol of arity n, and $t_1, \ldots, t_n$ are terms,
  then $f^n(t_1, \ldots, t_n)$ is a term.
  (The superscript may be omitted if no confusion results.)
- Nothing else is a term.

Atomic Formulas

If P^n is a predicate symbol of arity n,

and $t_1, \ldots, t_n$ are terms,

then $P^n(t_1, \ldots, t_n)$ is an atomic formula.

(The superscript may be omitted if no confusion results.)

Literals and Clauses

Literals: If P is an atomic formula,
then P and $\neg P$ are literals.

Clauses: If $L_1, \ldots, L_n$ are literals,
then the set $\{L_1, \ldots, L_n\}$ is a clause.

Sets of Clauses: If $C_1, \ldots, C_n$ are clauses,
then the set $\{C_1, \ldots, C_n\}$ is a set of clauses.

4.3.2 Semantics of Clause-Form First-Order Predicate Logic

- Individual Constants, Function Symbols, Predicate Symbols, Ground Terms, and Ground Atomic Formulas as for Standard FOL.
- Skolem Constants are like indefinite individuals.
- Skolem Function Symbols are like indefinite function symbols.
- Ground Literals, Ground Clauses, and Sets of Clauses as for Clause-Form Propositional Logic.

Semantics of Open Clauses

If clause C contains variables $v_1, \ldots, v_n$,
then $C\{t_1/v_1, \ldots, t_n/v_n\}$ is a **ground instance** of C if it contains no more variables.

If C is an open clause,
$[\![C]\!]$ is True if every ground instance of C is True.
Otherwise, it is False.

That is, variables take on **universal interpretation**,
with scope being the clause.

4.3.3 Proof Theory of Clause-Form FOL

Notion of Proof: None!

Notion of Derivation: A set of clauses constitutes a derivation.

Assumptions: The derivation is initialized with a set of assumption clauses $A_1, \ldots, A_n$.

Rule of Inference: A clause may be added to a set of clauses if justified by a rule of inference.

Derived Clause: If clause Q has been added to a set of clauses initialized with the set of assumption clauses $A_1, \ldots, A_n$ by one or more applications of resolution,
then $A_1, \ldots, A_n \vdash Q$.

Clause-Form FOL Rules of Inference
Version 1

Resolution: $$\frac{\{P, L_1, \ldots, L_n\}, \{\neg P, L_{n+1}, \ldots, L_m\}}{\{L_1, \ldots, L_n, L_{n+1}, \ldots, L_m\}}$$

Universal Instantion (temporary): $$\frac{C}{C\sigma}$$

Example Derivation

1.	$\{\neg Drives(x, y), Driver(x)\}$	$Assumption$
2.	$\{\neg Driver(z), \neg Passenger(z)\}$	$Assumption$
3.	$\{Drives(motherOf(Tom), Tom)\}$	$Assumption$
4.	$\{\neg Drives(motherOf(Tom), Tom),$ $Driver(motherOf(Tom))\}$	$UI, 1, \{motherOf(Tom)/x, Tom/y\}$
5.	$\{Driver(motherOf(Tom))\}$	$R, 3, 4$
6.	$\{\neg Driver(motherOf(Tom)),$ $\neg Passenger(motherOf(Tom))\}$	$UI, 2, \{motherOf(Tom)/z\}$
7.	$\{\neg Passenger(motherOf(Tom))\}$	$R, 5, 6$

Motivation for a Shortcut

$$\{P(x), L_1(x), \ldots, L_n(x)\} \quad \{\neg P(y), L_{n+1}(y), \ldots, L_m(y)\}$$

$$\downarrow \{a/x, a/y\} \qquad \downarrow \{a/x, a/y\}$$

$$\{P(a), L_1(a), \ldots, L_n(a)\} \quad \{\neg P(a), L_{n+1}(a), \ldots, L_m(a)\}$$

$$\{L_1(a), \ldots, L_n(a), L_{n+1}(a), \ldots, L_m(a)\}$$

Most General Unifier

A most general unifier (**mgu**), of atomic formulas $\mathcal{A}$ and $\mathcal{B}$

is a substitution, μ,

such that $\mathcal{A}\mu = \mathcal{B}\mu =$ a common instance of $\mathcal{A}$ and $\mathcal{B}$

and such that every other common instance of $\mathcal{A}$ and $\mathcal{B}$ is an instance of it.

I.e., $\mathcal{A}\mu = \mathcal{B}\mu =$ a most general common instance of $\mathcal{A}$ and $\mathcal{B}$.

Example:
Unifier of $P(a, x, y)$ and $P(u, b, v)$ is $\{a/u, b/x, c/y, c/v\}$
giving $P(a, b, c)$
But more general is $\{a/u, b/x, y/v\}$ giving $P(a, b, y)$

Clause-Form FOL Rules of Inference
Version 2

$$\{A, L_1, \ldots, L_n\}, \{\neg B, L_{n+1}, \ldots, L_m\}$$

Resolution: ---

$$\{L_1\mu, \ldots, L_n\mu, L_{n+1}\mu, \ldots, L_m\mu\}$$

where μ is an mgu of A and B.

Assume two parent clauses have no variables in common.

Example Derivation Revisited

1.	$\{\neg Drives(x, y), Driver(x)\}$	$Assumption$
2.	$\{\neg Driver(z), \neg Passenger(z)\}$	$Assumption$
3.	$\{Drives(motherOf(Tom), Tom)\}$	$Assumption$
4.	$\{Driver(motherOf(Tom))\}$	$R, 1, 3, \{motherOf(Tom)/x, Tom/y\}$
5.	$\{\neg Passenger(motherOf(Tom))\}$	$R, 2, 4, \{motherOf(Tom)/z, \}$

Unification

To find the mgu of $\mathcal{A}$ and $\mathcal{B}$.

Some Examples:

$\mathcal{A}$	$\mathcal{B}$	mgu	mgci
$P(a,b)$	$P(a,b)$	$\{\}$	$P(a,b)$
$P(a)$	$P(b)$	$FAIL$	
$P(a,x)$	$P(y,b)$	$\{a/y, b/x\}$	$P(a,b)$
$P(a,x)$	$P(y,g(y))$	$\{a/y, g(a)/x\}$	$P(a,g(a))$
$P(x,f(x))$	$P(y,y)$	$FAIL \quad (occurs\ check)$	

Substitution Composition

$P\sigma\tau = ((P\sigma)\tau) = P(\sigma \circ \tau)$

Let $\sigma = \{t_1/v_1, \ldots, t_n/v_n\}$

$\sigma \circ \tau = \{t_1\tau/v_1, \ldots, t_n\tau/v_n\} \uplus \tau$

$\sigma \uplus \tau = \sigma \cup \{t/v \mid (t/v \in \tau) \wedge v \notin \sigma\}$

E.g.: $\{x/y, y/z\} \circ \{u/y, v/w\} = \{x/y, u/z, v/w\}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))` `(P (f u) v v)`

μ = $\{\}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))` `(P (f u) v v)`

μ = {}

`...`[`x`] `(g x) (g (f a)))` `...`[`(f u)`] `v v)`

μ = {}$\circ$`{(f u)/x}`=`{(f u)/x}`

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

...[(g (f a))]) ...[v])

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

...[(g (f a))]) ...[v])

...[(g (f a))]) ...[(g (f u))])

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

...[(g (f a))]) ...[v])

...[(g (f a))]) ...[(g (f u))])

...[a]))) ...[u])))

Manual Unification Algorithm

(P x (g x) (g (f a))) (P (f u) v v)

μ = {}

...[x] (g x) (g (f a))) ...[(f u)] v v)

μ = {}∘{(f u)/x}={(f u)/x}

...[(g x)] (g (f a))) ...[v] v)

...[(g (f u))] (g (f a))) ...[v] v)

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

...[(g (f a))]) ...[v])

...[(g (f a))]) ...[(g (f u))])

...[a]))) ...[u])))

μ = {(f u)/x, (g (f u))/v}∘{a/u} = {(f a)/x, (g (f a))/v, a/u}

(P x (g x) (g (f a)))μ = (P (f u) v v)μ = (P (f a) (g (f a)) (g (f a)))

Unification Algorithm

```
(defun unify (A B &optional mu)
  (cond ((eql mu 'FAIL) 'FAIL)
        ((eql A B) mu)
        ((variablep A) (unifyVar A B mu))
        ((variablep B) (unifyVar B A mu))
        ((or (atom A) (atom B)) 'FAIL)
        ((/= (length A) (length B)) 'FAIL)
        (t (unify (rest A)
                  (rest B)
                  (unify (first A) (first B) mu)))))
```

Note: a more efficient version is implemented in `prover.cl`

UnifyVar

```
(defun unifyVar (var term subst)
  (if (var-in-substp var subst)
      (unify (term-of-var-in-subst var subst) term subst)
    (let ((newterm (apply-sub subst term)))
      (cond ((eql var newterm) subst)
            ((occursIn var newterm) 'FAIL)
            (t (compose subst
                        (list (pair newterm var))))))))
```

Program Assertion

If original $\mathcal{A}$ and $\mathcal{B}$ have no variables in common,

then throughout the above program

no substitution will have one of its variables occurring in one of its terms.

Therefore, for any expression $\mathcal{E}$ and any substitution σ formed in the above program, $\mathcal{E}\sigma\sigma = \mathcal{E}\sigma$.

4.3.4 Resolution Refutation Example

To decide if

$\{\neg Drives(x, y), Driver(x)\}, \{\neg Driver(x), \neg Passenger(x)\},$
$\{Drives(motherOf(Tom), Betty)\}$
$\models \{\neg Passenger(motherOf(Tom))\}$

1.	$\{\neg Drives(x_1, y_1), Driver(x_1)\}$	Assumption
2.	$\{\neg Driver(x_2), \neg Passenger(x_2)\}$	Assumption
3.	$\{Drives(motherOf(Tom), Betty)\}$	Assumption
5.	$\{Passenger(motherOf(Tom))\}$	From query
6.	$\{\neg Driver(motherOf(Tom))\}$	$R, 2, 5, \{motherOf(Tom)/x_2\}$
7.	$\{\neg Drives(motherOf(Tom), y_7)\}$	$R, 1, 6, \{motherOf(Tom)/x_1\}$
8.	$\{\}$	$R, 3, 7, \{Betty/y_7\}$

Example Using prover

```
prover(21): (prove '((or (not (Drives ?x ?y)) (Driver ?x))
                     (or (not (Driver ?x)) (not (Passenger ?x)))
                     (Drives (motherOf Tom) Betty))
                   '(not (Passenger (motherOf Tom))))

 1  ((Drives (motherOf Tom) Betty)) Assumption
 2  ((not (Drives ?3 ?5)) (Driver ?3))  Assumption
 3  ((not (Driver ?9)) (not (Passenger ?9)))  Assumption
 4  ((Passenger (motherOf Tom)))  From Query
 5  ((not (Driver (motherOf Tom)))) R,4,3,{(motherOf Tom)/?9}
 6  ((not (Drives (motherOf Tom) ?86))) R,5,2,{(motherOf Tom)/?3}
 7  nil              R,6,1,{Betty/?86}
QED
```

Example Using snark

```
snark-user(84): (initialize)
snark-user(85): (assert '(or (not (Drives ?x ?y)) (Driver ?x)))
snark-user(86): (assert '(or (not (Driver ?x))
                             (not (Passenger ?x))))
snark-user(87): (assert '(Drives (motherOf Tom) Betty))
snark-user(88): (prove '(not (Passenger (motherOf Tom))))
(Refutation
(Row 1 (or (not (Drives ?x ?y)) (Driver ?x)) assertion)
(Row 2 (or (not (Driver ?x)) (not (Passenger ?x))) assertion)
(Row 3 (Drives (motherOf Tom) Betty) assertion)
(Row 4 (Passenger (motherOf Tom)) ~conclusion)
(Row 5 (not (Driver (motherOf Tom))) (resolve 2 4))
(Row 6 (not (Drives (motherOf Tom) ?x)) (resolve 5 1))
(Row 7 false (resolve 6 3))
)
:proof-found
```

Resolution Refutation is Incomplete for FOL

1. $\{P(u), P(v)\}$
2. $\{\neg P(x), \neg P(y)\}$
3. $\{P(w), \neg P(z)\}$ $\quad R, 1, 2, \{u/x, w/v, z/y\}$

$\vdots$

Clause-Form FOL Rules of Inference
Version 3 (Last)

Resolution: $$\frac{\{A, L_1, \ldots, L_n\}, \{\neg B, L_{n+1}, \ldots, L_m\}}{\{L_1\mu, \ldots, L_n\mu, L_{n+1}\mu, \ldots, L_m\mu\}}$$

where μ is an mgu of A and B.

Factoring: $$\frac{\{A, B, L_1, \ldots, L_n\}}{\{A\mu, L_1\mu, \ldots, L_n\mu\}}$$

where μ is an mgu of A and B.
(Note: Special case of UI.)

Resolution Refutation with Factoring is Complete

If $A_1, \ldots, A_n \models Q$, then $A_1, \ldots, A_n, \neg Q \vdash_{R+F} \{\}$.

For example,

1.	$\{P(u), P(v)\}$	
2.	$\{\neg P(x), \neg P(y)\}$	
3.	$\{P(w)\}$	$F, 1, \{w/u, w/v\}$
4.	$\{\neg P(z)\}$	$F, 2, \{z/x, z/y\}$
5.	$\{\}$	$R, 3, 4, \{w/z\}$

However, resolution refutation with factoring is still not a decision procedure—it is a semi-decision procedure.

Factoring (Condensing) by snark

```
snark-user(30): (initialize)
; Running SNARK from ...
nil
snark-user(31): (assert '(or (P ?u) (P ?v)))
nil
snark-user(32): (prove '(and (P ?x) (P ?y)))

(Refutation
(Row 1
   (or (P ?x) (P ?y))
   assertion)
(Row 2
   (P ?x)
   (condense 1))
(Row 3
   (or (not (P ?x)) (not (P ?y)))
   negated_conjecture)
(Row 4
   false
   (rewrite 3 2))
)

:proof-found
```

SNARK has both factoring and condensing, which is factoring combined with immediate subsumption elimination when the factored clause subsumes the original clause. The clause '(or (p a ?x) (p ?y b)) gets factored, but not condensed. [Mark Stickel, personal communication, March, 2008]

Efficiency Rules

Tautology Elimination: If clause C contains literals L and $\neg L$, delete C from the set of clauses. (Check throughout.)

Pure-Literal Elimination: If clause C contains a literal A ($\neg A$) and no clause contains a literal $\neg B$ (B) such that A and B are unifiable, delete C from the set of clauses. (Check throughout.)

Subsumption Elimination: If the set of clauses contains clauses C_1 and C_2 such that there is a substitution σ for which $C_1\sigma \subseteq C_2$, delete C_2 from the set of clauses. (Check throughout.)

These rules delete unhelpful clauses.

Subsumption may be required to cut infinite loops.

Subsumption Cutting a Loop

```
prover(22): (prove '((if (and (ancestor ?x ?y)
                              (ancestor ?y ?z))
                         (ancestor ?x ?z)))
                   '(ancestor ?x stu))

 1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
     (ancestor ?0 ?2))  Assumption
 2  ((not (ancestor ?3 stu))) From Query
```

Initial Resolution Steps

```
1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
    (ancestor ?0 ?2))                                    Assumption
2  ((not (ancestor ?3 stu)))                             From Query
3  ((not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
                                          R,2,1,{stu/?2, ?0/?3}
4  ((not (ancestor ?6 stu)) (not (ancestor ?7 ?8))
    (not (ancestor ?8 ?6)))                R,3,1,{?2/?5, ?0/?4}
5  ((not (ancestor ?9 ?10)) (not (ancestor ?10 ?11))
    (not (ancestor ?11 stu)))                R,3,1,{stu/?2, ?0/?5}
                .
                .
                .
```

Subsumption Cuts the Loop

```
 1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
     (ancestor ?0 ?2))                                  Assumption
 2  ((not (ancestor ?3 stu)))                          From Query
 3  (not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
                                              R,2,1,{stu/?2, ?0/?3}
 4  ((not (ancestor stu stu)))  F,3,{stu/?5, stu/?4}
Deleting  4 ((not (ancestor stu stu)))
 because it's subsumed by  2 ((not (ancestor ?3 stu)))
Deleting  3 ((not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
 because it's subsumed by  2 ((not (ancestor ?3 stu)))
nil
```

Strategies

Unit Preference: Resolve shorter clauses before longer clauses.
Least Symbol Count Version: Count symbols, not literals.

Set of Support: One clause in each pair being resolved must descend from the query.

Many others

These are heuristics for finding {} faster.

Least Symbol Count Version of Unit Preference

Instead of counting literals,

count symbols

ignoring negation operator.

Equivalent to standard unit preference for Propositional Logic.

Problem with
Literal-Counting Unit Preference

```
1(1/2)  ((walkslikeduck daffy)) Assumption
2(1/2)  ((talkslikeduck daffy)) Assumption
3(2/5)  ((not (duck (motherof ?1))) (duck ?1))  Assumption
4(3/6)  ((not (walkslikeduck ?3)) (not (talkslikeduck ?3)) (duck ?3)) Assumption
5(1/2)  ((not (duck daffy)))  From Query
6(1/3)  ((not (duck (motherof daffy)))) R,5,3,{daffy/?1}
7(1/4)  ((not (duck (motherof (motherof daffy))))) R,6,3,{(motherof daffy)/?1}
8(1/5)  ((not (duck
               (motherof
                (motherof
                 (motherof daffy))))))  R,7,3,{(motherof (motherof daffy))/?1}
9(1/6)  ((not (duck
               (motherof
                (motherof
                 (motherof
                  (motherof
                   daffy)))))))  R,8,3,{(motherof (motherof (motherof daffy)))/?1}
```

Solution with
Least Symbol Count Version

```
 1(1/2)  ((walkslikeduck daffy)) Assumption
 2(1/2)  ((talkslikeduck daffy)) Assumption
 3(2/5)  ((not (duck (motherof ?5))) (duck ?5))  Assumption
 4(3/6)  ((not (walkslikeduck ?13)) (not (talkslikeduck ?13)) (duck ?13)) Assumption
 5(1/2)  ((not (duck daffy)))  From Query

 6(1/3)  ((not (duck (motherof daffy)))) R,5,3,{daffy/?1}
 7(1/4)  ((not (duck (motherof (motherof daffy))))) R,6,3,{(motherof daffy)/?1}
 8(2/4)  ((not (walkslikeduck daffy)) (not (talkslikeduck daffy))) R,5,4,{daffy/?3}
 9(1/2)  ((not (talkslikeduck daffy))) R,8,1,{}
10(0/0)  nil             R,9,2,{}
QED
```

4.4 Translating Standard FOL Wffs into FOL Clause Form

Useful Meta-Theorems

- If A is (an occurrence of) a subformula of B,
  and $\models A \Leftrightarrow C$,
  then $\models B \Leftrightarrow B\{C/A\}$

- $\forall x_1(\cdots \forall x_n(\cdots \exists y A(x_1, \ldots, x_n, y) \cdots) \cdots)$ is consistent
  if and only if
  $\forall x_1(\cdots \forall x_n(\cdots A(x_1, \ldots, x_n, f^n(x_1, \ldots, x_n)) \cdots) \cdots)$
  is consistent,
  where f^n is a new Skolem function.
  Note: use a new Skolem constant instead of $f^0()$.

Translating Standard FOL Wffs into FOL Clause Form

Step 1

Eliminate occurrences of $\Leftrightarrow$ using

$$\models (A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$$

From:

$\forall x[Parent(x) \Leftrightarrow (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x)))]$

To:

$\forall x[(Parent(x) \Rightarrow (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))))$
$\wedge((Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))) \Rightarrow Parent(x))]$

Translation Step 2

Eliminate occurrences of $\Rightarrow$ using

$$\models (A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$$

From:

$\forall x[(Parent(x) \Rightarrow (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))))$
$\quad \wedge((Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))) \Rightarrow Parent(x))]$

To:

$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))))$
$\quad \wedge(\neg(Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))) \vee Parent(x))]$

Translation Step 3

Translate to *miniscope* form using

$$\models \neg\neg A \Leftrightarrow A$$

$$\models \neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B) \qquad \models \neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$$

$$\models \neg\forall x A(x) \Leftrightarrow \exists x \neg A(x) \qquad \models \neg\exists x A(x) \Leftrightarrow \forall x \neg A(x)$$

From:

$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$
$\quad \wedge(\neg(Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))) \vee Parent(x))]$

To:

$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$
$\quad \wedge((\neg Person(x) \vee \forall y(\neg Person(y) \vee \neg childOf(y,x))) \vee Parent(x))]$

Translation Step 4

Rename apart: If any two quantifiers bind the same variable, rename all occurrences of one of them.

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee \forall y(\neg Person(y) \vee \neg childOf(y,x))) \vee Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

Optional Translation Step 4.5

Translate into Prenex Normal Form using:

$$\models (A \wedge \forall x B(x)) \Leftrightarrow \forall x(A \wedge B(x)) \quad \models (A \wedge \exists x B(x)) \Leftrightarrow \exists x(A \wedge B(x))$$

$$\models (A \vee \forall x B(x)) \Leftrightarrow \forall x(A \vee B(x)) \quad \models (A \vee \exists x B(x)) \Leftrightarrow \exists x(A \vee B(x))$$

as long as x does not occur free in A.

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

To:

$$\forall x \exists y \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(y) \wedge childOf(y, x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

Translation Step 5

Skolemize

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x),x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

or

From:

$$\forall x \exists y \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

To:

$$\forall x \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x),x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

Translation Step 6

Discard all occurrences of "$\forall x$" for any variable x.

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

Or from:

$$\forall x \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

To:

$$[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

Translation Step 7

CNF: Translate into Conjunctive Normal Form, using

$$\models (A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$$

$$\models ((B \wedge C) \vee A) \Leftrightarrow ((B \vee A) \wedge (C \vee A))$$

From:

$[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$
$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$

To:

$[((\neg Parent(x) \vee Person(x))$
$\wedge((\neg Parent(x) \vee Person(f(x)))$
$\wedge(\neg Parent(x) \vee childOf(f(x), x)))))$
$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$

Translation Step 8

Discard extra parentheses using the associativity of $\wedge$ and $\vee$.

From:

$$[((\neg Parent(x) \vee Person(x))$$
$$\wedge((\neg Parent(x) \vee Person(f(x)))$$
$$\wedge(\neg Parent(x) \vee childOf(f(x), x)))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

To:

$$[(\neg Parent(x) \vee Person(x))$$
$$\wedge(\neg Parent(x) \vee Person(f(x)))$$
$$\wedge(\neg Parent(x) \vee childOf(f(x), x))$$
$$\wedge(\neg Person(x) \vee \neg Person(z) \vee \neg childOf(z, x) \vee Parent(x))]$$

Translation Step 9

Turn each disjunction into a clause,
and the conjunction into a set of clauses.

From:

$$[(\neg Parent(x) \vee Person(x))$$
$$\wedge(\neg Parent(x) \vee Person(f(x)))$$
$$\wedge(\neg Parent(x) \vee childOf(f(x), x))$$
$$\wedge(\neg Person(x) \vee \neg Person(z) \vee \neg childOf(z, x) \vee Parent(x))]$$

To:

$$\{\{\neg Parent(x), Person(x)\},$$
$$\{\neg Parent(x), Person(f(x))\},$$
$$\{\neg Parent(x), childOf(f(x), x)\},$$
$$\{\neg Person(x), \neg Person(z), \neg childOf(z, x), Parent(x)\}\}$$

Translation Step 10

Rename the clauses apart
so that no variable occurs in more than one clause.

From:
$\{\{\neg Parent(x), Person(x)\},$
$\{\neg Parent(x), Person(f(x))\},$
$\{\neg Parent(x), childOf(f(x), x)\},$
$\{\neg Person(x), \neg Person(z), \neg childOf(z, x), Parent(x)\}\}$
To:
$\{\{\neg Parent(x_1), Person(x_1)\},$
$\{\neg Parent(x_2), Person(f(x_2))\},$
$\{\neg Parent(x_3), childOf(f(x_3), x_3)\},$
$\{\neg Person(x_4), \neg Person(z_4), \neg childOf(z_4, x_4), Parent(x_4)\}\}$

Use of Translation

$$A_1, \ldots, A_n \models B$$

iff

The translation of $A_1 \wedge \cdots \wedge A_n \wedge \neg B$ into a set of clauses is contradictory.

Example with ubprover

```
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(Parent Betty))

 1  ((Person Tom))                                  Assumption
 2  ((Person Betty))                                Assumption
 3  ((childOf Tom Betty))                           Assumption
 4  ((not (Parent ?4)) (Person ?4))                 Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))            Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))        Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))             Assumption
 8  ((not (Parent Betty)))                          From Query
```

Resolution Steps

```
 1  ((Person Tom))                                    Assumption
 2  ((Person Betty))                                  Assumption
 3  ((childOf Tom Betty))                             Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))               Assumption
 8  ((not (Parent Betty)))                            From Query
 9  ((not (Person Betty)) (not (Person ?9))
     (not (childOf ?9 Betty)))                        R,8,7,{Betty/?7}
13  ((not (Person Betty))
     (not (childOf Tom Betty)))                       R,9,1,{Tom/?9}
14  ((not (childOf Tom Betty)))                       R,13,2,{}
15  nil                                               R,14,3,{}
QED
```

Example with SNARK

```
snark-user(42): (initialize)
; Running SNARK from ...
nil
snark-user(43): (assert
                 '(forall (x)
                   (iff (Parent x)
                        (and (Person x)
                             (exists (y)
                              (and (Person y) (childOf y x)))))))
nil
snark-user(44): (assert '(Person Tom))
nil
snark-user(45): (assert '(Person Betty))
nil
snark-user(46): (assert '(childOf Tom Betty))
nil
snark-user(47): (prove '(Parent Betty))
```

Initial Set of Clauses

```
(Row 1 (or (not (Parent ?x)) (Person ?x)) assertion)
(Row 2 (or (not (Parent ?x)) (Person (skolembiry1 ?x))) assertion)
(Row 3 (or (not (Parent ?x)) (childOf (skolembiry1 ?x) ?x)) assertion)
(Row 4 (or (Parent ?x) (not (Person ?x)) (not (Person ?y)) (not (childOf ?y ?
       assertion)
(Row 5 (Person Tom) assertion)
(Row 6 (Person Betty) assertion)
(Row 7 (childOf Tom Betty) assertion)
(Row 8 (not (Parent Betty)) negated_conjecture)
(Row 9 (or (not (Person ?x)) (not (childOf ?x Betty))) (rewrite (resolve 8 4)
(Row 10 false (rewrite (resolve 9 7) 5))
```

Refutation

```
(Refutation
(Row 4 (or (Parent ?x) (not (Person ?x)) (not (Person ?y)) (not (childOf ?y ?
       assertion)
(Row 5 (Person Tom) assertion)
(Row 6 (Person Betty) assertion)
(Row 7 (childOf Tom Betty) assertion)
(Row 8 (not (Parent Betty)) negated_conjecture)
(Row 9 (or (not (Person ?x)) (not (childOf ?x Betty))) (rewrite (resolve 8 4)
(Row 10 false (rewrite (resolve 9 7) 5))
)
:proof-found
```

A ubprover Example
Using the Skolem Function

```
prover(72): (prove
             '((forall x (iff (Parent x)
                              (and (Person x)
                                   (exists y (and (Person y) (childOf y x))))))
               (Person Tom) (Person Betty) (Parent Betty))
             '(exists x (childOf x Betty)))

 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((Parent Betty))                            Assumption
 4  ((not (Parent ?4)) (Person ?4))             Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))        Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))    Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))         Assumption
 8  ((not (childOf ?10 Betty)))                 From Query
 9  ((not (Parent Betty)))                      R,8,6,{Betty/?6, (S3 Betty)/?10}
10  nil                                         R,9,3,{}
QED
```

4.5 Asking Wh Questions

Given

$\forall x[Parent(x) \Leftrightarrow (Person(x) \wedge \exists y(Person(y) \wedge childOf(y, x)))]$

$Person(Tom)$

$Person(Betty)$

$childOf(Tom, Betty)$

Ask: "Who is a parent?"

Answer via constructive proof of $\exists x\ Parent(x)$.

Try to Answer Wh Question

```
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(exists x (Parent x)))

 1  ((Person Tom))                                  Assumption
 2  ((Person Betty))                                Assumption
 3  ((Parent Betty))                                Assumption
 4  ((not (Parent ?4)) (Person ?4))                 Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))            Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))        Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))             Assumption
 8  ((not (childOf ?10 Betty)))                     From Query
```

Resolution Steps

```
 1  ((Person Tom))                                          Assumption
 2  ((Person Betty))                                        Assumption
 3  ((childOf Tom Betty))                                   Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))                     Assumption
 8  ((not (Parent ?10)))                                    From Query
 9  ((not (Person ?11)) (not (Person ?12))
     (not (childOf ?12 ?11)))                               R,8,7,{?7/?10}
15  ((not (Person ?16)) (not (childOf Tom ?16)))            R,9,1,{Tom/?12}
16  ((not (childOf Tom Tom)))                               R,15,1,{Tom/?16}
17  ((not (childOf Tom Betty)))                             R,15,2,{Betty/?16}
18  nil                                                     R,17,3,{}
QED
```

The Answer Predicate

Instead of query $\exists x_1 \cdots \exists x_n P(x_1, \ldots, x_n)$,

and resolution refutation with $\{\neg P(x_1, \ldots, x_n)\}$

until $\{\}$,

use $\forall x_1 \cdots \forall x_n (P(x_1, \ldots, x_n) \Rightarrow Answer(P(x_1, \ldots, x_n)))$

and do direct resolution with

$$\{\neg P(x_1, \ldots, x_n), Answer(P(x_1, \ldots, x_n))\}$$

until $\{(Answer \ldots) \cdots (Answer \ldots)\}$.

General Procedure for Inserting The Answer Predicate

Let:

Q be either $\forall$ or $\exists$;

$\overline{Q}$ be either $\exists$ or $\forall$, respectively;

Prenex Normal form of query be $Q_1 x_1 \cdots Q_n x_n P(x_1, \ldots, x_n)$.

Do direct resolution with clause form of

$\overline{Q_1} x_1 \cdots \overline{Q_n} x_n (P(x_1, \ldots, x_n) \Rightarrow Answer(P(x_1, \ldots, x_n)))$

until generate $\{(Answer \ldots) \cdots (Answer \ldots)\}$.

Using the Answer Predicate

```
(setf *UseAnswer* t)
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(exists x (Parent x)))

 1  ((Person Tom))                                Assumption
 2  ((Person Betty))                              Assumption
 3  ((childOf Tom Betty))                         Assumption
 4  ((not (Parent ?3)) (Person ?3))               Assumption
 5  ((not (Parent ?4)) (Person (S2 ?4)))          Assumption
 6  ((not (Parent ?5)) (childOf (S2 ?5) ?5))      Assumption
 7  ((not (Person ?6)) (not (Person ?7))
     (not (childOf ?7 ?6)) (Parent ?6))           Assumption
 8  ((not (Parent ?9)) (Answer (Parent ?9)))      From Query
```

Resolution Steps

```
 1  ((Person Tom))                                          Assumption
 2  ((Person Betty))                                        Assumption
 3  ((childOf Tom Betty))                                   Assumption
 7  ((not (Person ?6)) (not (Person ?7))
     (not (childOf ?7 ?6)) (Parent ?6))                     Assumption
 8  ((not (Parent ?9)) (Answer (Parent ?9)))                From Query
 9  ((Answer (Parent ?10)) (not (Person ?10))
     (not (Person ?11)) (not (childOf ?11 ?10)))            R,8,7,{?6/?9}
15  ((Answer (Parent Betty))
     (not (Person Betty)) (not (Person Tom)))               R,9,3,{Betty/?10,
                                                                   Tom/?11}
26  ((Answer (Parent Betty)) (not (Person Tom)))            R,15,2,{}
29  ((Answer (Parent Betty)))                               R,26,1,{}
QED
```

Answer Predicate in snark

```
snark-user(11): (assert '(forall x (iff (Parent x)
                                        (exists y (and (Person y)
                                                       (childOf y x))))))
nil
snark-user(12): (assert '(Person Tom))
nil
snark-user(13): (assert '(Person Betty))
nil
snark-user(14): (assert '(childOf Tom Betty))
nil
snark-user(15): (prove '(exists x (Parent x))
                       :answer '(Parent x))
```

snark Refutation

```
(Refutation
(Row 3
   (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (childOf Tom Betty) assertion)
(Row 7 (not (Parent ?x)) negated_conjecture
   Answer (Parent ?x))
(Row 8 (or (not (Person ?x)) (not (childOf ?x ?y))) (resolve 7 3)
   Answer (Parent ?y))
(Row 9 false (rewrite (resolve 8 6) 4)
   Answer (Parent Betty))
)
:proof-found
```

Answer Predicate with `ask`

From same SNARK KB:

```
snark-user(18): (ask '(exists x (Parent x)) :answer '(Parent x))
(Parent Betty)
```

Using :printProof

```
snark-user(19): (ask '(Parent ?x) :answer '(Parent ?x)
                                  :printProof t)
(Refutation
(Row 3 (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (childOf Tom Betty) assertion)
(Row 13 (not (Parent ?x)) negated_conjecture
   Answer (Parent ?x))
(Row 14 (or (not (Person ?x)) (not (childOf ?x ?y)))
   (resolve 13 3)
   Answer (Parent ?y))
(Row 15 false (rewrite (resolve 14 6) 4)
   Answer (Parent Betty))
)
(Parent Betty)
```

Answer Predicate with `query`

From same SNARK KB:

```
snark-user(9): (query "Who is a parent?"
                      '(exists x (Parent x))
                      :answer '(Parent x))

Who is a parent?
  (ask '(exists x (Parent x))) = (Parent Betty)
```

query with :answer and :printProof

```
snark-user(10): (query "Who is a parent?"
'(exists x (Parent x)) :answer '(Parent x) :printProof t)
Who is a parent?
(Refutation
(Row 3
   (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4
   (Person Tom)
   assertion)
(Row 6
   (childOf Tom Betty)
   assertion)
(Row 19
   (not (Parent ?x))
   negated_conjecture
   Answer (Parent ?x))
(Row 20
   (or (not (Person ?x)) (not (childOf ?x ?y)))
   (resolve 19 3)
   Answer (Parent ?y))
(Row 21
   false
   (rewrite (resolve 20 6) 4)
   Answer (Parent Betty))
)
  (ask '(exists x (Parent x))) = (Parent Betty)
```

Disjunctive Answers

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         (or (Red b) (Green b)))
       '(exists (x y)
                (and (Red x) (Green y) (On x y))))
 1  ((On a b))                                                    Assumption
 2  ((On b c))                                                    Assumption
 3  ((Red a))                                                     Assumption
 4  ((Green c))                                                   Assumption
 5  ((Red b) (Green b))                                           Assumption
 6  ((not (Red ?28)) (not (Green ?30))
     (not (On ?28 ?30))
     (Answer (and (Red ?28) (Green ?30) (On ?28 ?30)))) From Query
```

Resolution Steps

```
 9  ((Answer (and (Red a) (Green ?107) (On a ?107)))
     (not (On a ?107)) (not (Green ?107)))                    R,6,3,{a/?28}

10  ((Answer (and (Red ?112) (Green c) (On ?112 c)))
     (not (On ?112 c)) (not (Red ?112)))                      R,6,4,{c/?30}

11  ((Answer (and (Red b) (Green ?117) (On b ?117)))
     (not (On b ?117)) (not (Green ?117)) (Green b))          R,6,5,{b/?28}

13  ((not (Red b))
     (Answer (and (Red b) (Green c) (On b c))))               R,10,2,{b/?112}
```

Resolution Finished

```
16  ((Answer (and (Red b) (Green c) (On b c)))
     (Green b))                                          R,13,5,{}

20  ((not (On a b))
     (Answer (and (Red a) (Green b) (On a b)))
     (Answer (and (Red b) (Green c) (On b c)))) R,9,16,{b/?107}

22  ((Answer (and (Red b) (Green c) (On b c)))
     (Answer (and (Red a) (Green b) (On a b)))) R,20,1,{}
QED
```

Multiple Clauses From Query

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         (or (Red b) (Green b)))
       '(exists x (or (Red x) (Green x))))

 1  ((On a b))                                  Assumption
 2  ((On b c))                                  Assumption
 3  ((Red a))                                   Assumption
 4  ((Green c))                                 Assumption
 5  ((Red b) (Green b))                         Assumption
 6  ((not (Red ?25))
     (Answer (or (Red ?25) (Green ?25))))       From Query
 7  ((not (Green ?27))
     (Answer (or (Red ?27) (Green ?27))))       From Query
 8  ((Answer (or (Red a) (Green a))))           R,6,3,{a/?25}
QED
```

Resolution Produces Only 1 Answer

```
snark-user(20): (initialize)
; Running SNARK from ...
nil

snark-user(21): (assert '(Man Socrates))
nil

snark-user(22): (assert '(Man Turing))
nil

snark-user(23): (ask '(Man ?x) :answer '(One man is ?x))
(One man is Turing)
```

Generic and Hypothetical Answers

Every clause that descends from a query clause (that contains an `Answer` predicate) is an answer of some sort.[a]

[a]Debra T. Burhans and Stuart C. Shapiro, Defining Answer Classes Using Resolution Refutation, *Journal of Applied Logic 5*, 1 (March 2007), 70–91 . `http://www.cse.buffalo.edu/~shapiro/Papers/bursha05.pdf`

Example of Generic and Hypothetical Answers Question

```
(prove '((forall (x y z) (if (and (Member x FBS) (Sport y)
                                  (Athlete z) (PlaysWell z y))
                             (ProvidesScholarshipFor x z)))
         (forall x (if (Sport x) (Activity x)))
         (forall x (if (Activity x) (or (Sport x) (Game x))))
         (forall x (if (or (Member x MAC) (Member x Big10) (Member Pac10 x))
                       (Member x FBS)))
         (Member Buffalo MAC) (Member KentSt MAC)
         (Member Wisconsin Big10) (Member Indiana Big10)
         (Member Stanford Pac10) (Member Berkeley Pac10)
         (Activity Frisbee))

        '(exists x (ProvidesScholarshipFor Buffalo x)))
```

Initial Clauses

```
 1  ((Member Buffalo MAC))                                          Assumption
 2  ((Member KentSt MAC))                                           Assumption
 3  ((Member Wisconsin Big10))                                      Assumption
 4  ((Member Indiana Big10))                                        Assumption
 5  ((Member Stanford Pac10))                                       Assumption
 6  ((Member Berkeley Pac10))                                       Assumption
 7  ((Activity Frisbee))                                            Assumption
 8  ((not (Sport ?7)) (Activity ?7))                                Assumption
 9  ((not (Member ?11 MAC)) (Member ?11 FBS))                       Assumption
10  ((not (Member ?12 Big10)) (Member ?12 FBS))                     Assumption
11  ((not (Member Pac10 ?13)) (Member ?13 FBS))                     Assumption
12  ((not (Activity ?9)) (Sport ?9) (Game ?9))                      Assumption
13  ((not (Member ?3 FBS)) (not (Sport ?4)) (not (Athlete ?5))
     (not (PlaysWell ?5 ?4)) (ProvidesScholarshipFor ?3 ?5))        Assumption
14  ((not (ProvidesScholarshipFor Buffalo ?15))
     (Answer (ProvidesScholarshipFor Buffalo ?15)))                 From Query
```

Resolvents

```
15  ((Answer (ProvidesScholarshipFor Buffalo ?16)) (not (Member Buffalo FBS)) (not (Sport ?17))
     (not (Athlete ?16)) (not (PlaysWell ?16 ?17)))  R,14,13,{?5/?15, Buffalo/?3}
16  ((not (Member Buffalo MAC)) (Answer (ProvidesScholarshipFor Buffalo ?18)) (not (Sport ?19))
     (not (Athlete ?18)) (not (PlaysWell ?18 ?19)))  R,15,9,{Buffalo/?11}
17  ((not (Member Buffalo Big10)) (Answer (ProvidesScholarshipFor Buffalo ?20)) (not (Sport ?21))
     (not (Athlete ?20)) (not (PlaysWell ?20 ?21)))  R,15,10,{Buffalo/?12}
18  ((not (Member Pac10 Buffalo)) (Answer (ProvidesScholarshipFor Buffalo ?22)) (not (Sport ?23))
     (not (Athlete ?22)) (not (PlaysWell ?22 ?23)))  R,15,11,{Buffalo/?13}
19  ((Game ?24) (not (Activity ?24)) (Answer (ProvidesScholarshipFor Buffalo ?25))
     (not (Member Buffalo FBS)) (not (Athlete ?25)) (not (PlaysWell ?25 ?24)))  R,15,12,{?9/?17}
20  ((Game ?26) (not (Activity ?26)) (not (Member Pac10 Buffalo)) (Answer (ProvidesScholarshipFor Buffalo ?27))
     (not (Athlete ?27)) (not (PlaysWell ?27 ?26)))  R,18,12,{?9/?23}
21  ((Game ?28) (not (Activity ?28)) (not (Member Buffalo Big10)) (Answer (ProvidesScholarshipFor Buffalo ?29))
     (not (Athlete ?29)) (not (PlaysWell ?29 ?28)))  R,17,12,{?9/?21}
22  ((Answer (ProvidesScholarshipFor Buffalo ?30)) (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31))) R,16,1,{}
23  ((Game ?32) (not (Activity ?32)) (not (Member Buffalo MAC)) (Answer (ProvidesScholarshipFor Buffalo ?33))
     (not (Athlete ?33)) (not (PlaysWell ?33 ?32)))  R,16,12,{?9/?19}
24  ((Game ?34) (not (Activity ?34)) (Answer (ProvidesScholarshipFor Buffalo ?35)) (not (Athlete ?35))
     (not (PlaysWell ?35 ?34))) R,22,12,{?9/?31}
25  ((Game Frisbee) (Answer (ProvidesScholarshipFor Buffalo ?36)) (not (Athlete ?36))
     (not (PlaysWell ?36 Frisbee)))  R,24,7,{Frisbee/?34}
26  ((not (Sport ?37)) (Game ?37) (Answer (ProvidesScholarshipFor Buffalo ?38)) (not (Athlete ?38))
     (not (PlaysWell ?38 ?37)))  R,24,8,{?7/?34}
nil
```

Non-Subsumed Resolvents

```
22  ((Answer (ProvidesScholarshipFor Buffalo ?30))
     (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31)))

24  ((Game ?34) (not (Activity ?34))
     (Answer (ProvidesScholarshipFor Buffalo ?35))
     (not (Athlete ?35)) (not (PlaysWell ?35 ?34)))

25  ((Game Frisbee)
     (Answer (ProvidesScholarshipFor Buffalo ?36))
     (not (Athlete ?36)) (not (PlaysWell ?36 Frisbee)))
```

Interpretation of Clauses As Generic Answers

```
22  ((Answer (ProvidesScholarshipFor Buffalo ?30))
     (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31)))
```

$$\forall xy[Athlete(x) \wedge Sport(y) \wedge PlaysWell(x, y) \Rightarrow ProvidesScholarshipFor(Buffalo, x)]$$

```
24  ((Game ?34) (not (Activity ?34))
     (Answer (ProvidesScholarshipFor Buffalo ?35))
     (not (Athlete ?35)) (not (PlaysWell ?35 ?34)))
```

$$\forall xy[Athlete(x) \wedge Activity(y) \wedge \neg Game(y) \wedge PlaysWell(x, y) \Rightarrow ProvidesScholarshipFor(Buffalo, x)]$$

Interpretation of Clause As Hypothetical Answer

```
25  ((Game Frisbee)
     (Answer (ProvidesScholarshipFor Buffalo ?36))
     (not (Athlete ?36)) (not (PlaysWell ?36 Frisbee)))
```

$$\neg Game(Frisbee) \Rightarrow \forall xy[Athlete(x) \wedge PlaysWell(x, Frisbee) \Rightarrow ProvidesScholarshipFor(Buffalo, x)]$$

Rule-Based Systems

Every FOL KB

can be expressed as a set of rules of the form

$\forall \overline{x}(C_1(\overline{x}) \vee \cdots \vee C_m(\overline{x}))$

or

$\forall \overline{x}(A_1(\overline{x}) \wedge \cdots \wedge A_n(\overline{x}) \Rightarrow C_1(\overline{x}) \vee \cdots \vee C_m(\overline{x}))$

or

$\forall \overline{x}(A_1(\overline{x}) \wedge \cdots \wedge A_n(\overline{x}) \Rightarrow C(\overline{x}))$

where $A_i(\overline{x})$ and $C_j(\overline{x})$ are literals.

Wh Questions in Rule-Based Systems

Given rule $\forall \overline{x}(A(\overline{x}) \Rightarrow C(\overline{x}))$

Ask $C(\overline{y})$?

Backchain to subgoal $A(\overline{x})\mu$, where μ is an mgu of $C(\overline{x}))$ and $C(\overline{y}))$

Moral: Unification is generally needed in backward chaining systems.

Unification is correct pattern matching when both structures have variables.

Forward Chaining & Unification

Forward chaining generally matches a ground fact with rule antecedents.

Forward chaining does not generally require unification.

Common Formalizing Difficulties

Every raven is black: $\forall x(Raven(x) \Rightarrow Black(x))$

Some raven is black: $\exists x(Raven(x) \wedge Black(x))$

Note the satisfying models of the incorrect
$\exists x(Raven(x) \Rightarrow Black(x))$

Another Formalizing Difficulty

Note where a Skolem function appears in

$\forall x(Parent(x) \Leftrightarrow \exists y childOf(y, x))$

$$\Leftrightarrow \forall x((Parent(x) \Rightarrow \exists y childOf(y, x)) \wedge((\exists y childOf(y, x)) \Rightarrow Parent(x)))$$

$$\Leftrightarrow \forall x((\neg Parent(x) \vee \exists y childOf(y, x)) \wedge(\neg(\exists y childOf(y, x)) \vee Parent(x)))$$

$$\Leftrightarrow \forall x((\neg Parent(x) \vee \exists y childOf(y, x)) \wedge(\forall y(\neg childOf(y, x)) \vee Parent(x)))$$

$$\Leftrightarrow \forall x(Parent(x) \Rightarrow childOf(f(x), x)) \wedge \forall x \forall y(childOf(y, x) \Rightarrow Parent(x))$$

What's "First-Order" About FOL?

In a first-order logic:

Predicate and function symbols cannot be arguments of predicates or functions;

Variables cannot be in predicate or function position.

E.G. $\forall r[\mathit{Transitive}(r) \Leftrightarrow \forall xyz[r(x,y) \wedge r(y,z) \Rightarrow r(x,z)]]$
is not a first-order sentence.

"The adjective 'first-order' is used to distinguish the languages we shall study here from those in which there are predicates having other predicates or functions as arguments or in which predicate quantifiers or function quantifiers are permitted, or both." [Elliott Mendelson, *Introduction to Mathematical Logic, Fifth Edition,* CRC Press, Boca Raton, FL, 2010, p. 48.]

Russell's Theory of Types

Designed to solve paradox: $\exists s \forall c[s(c) \Leftrightarrow \neg c(c)]$

has instance $S(S) \Leftrightarrow \neg S(S)$

N^{th}-Order Logic

Assign type 0 to individuals and to terms denoting individuals.

Assign type $i + 1$ to any set and to any function or predicate symbol that denotes a set, possibly of tuples, such that the maximum type of any of its elements is i.

Also assign type $i + 1$ to any variable that range over type i objects.

Note that the type of a functional term is the type of its range—the n^{th} element of the n-tuples of the set which the function denotes.

Syntactically, if the maximum type of the arguments of a ground atomic wff is i, then the type of the predicate is $i + 1$.
No predicate of type i may take a ground argument of type i or higher.

First-Order Logic Defined

First-order logic has a language that obeys Russell's Theory of Types, and whose highest type symbol is of type 1.

n^{th}-order logic has a language that obeys Russell's Theory of Types, and whose highest type symbol is of type n.

Ω-ordered logic has no limit, but must still follow the rules.

E.g., $\forall r[\mathit{Transitive}(r) \Leftrightarrow \forall xyz[r(x,y) \wedge r(y,z) \Rightarrow r(x,z)]]$
is a formula of Second-Order Logic:
Type 0 objects: individuals in the domain
Type 1 symbols: x, y, z because they range over type 0 objects
Type 1 objects: binary relations over type 0 objects
Type 2 symbols: r because it ranges over type 1 objects,
Transitive because it denotes a set of type 1 objects

Nested Beliefs

Can a proposition be an argument of a proposition?

Consider:

$$\forall p(Believes(Solomon, p) \Rightarrow p)$$
$$Believes(Solomon, Round(Earth)) \Rightarrow Round(Earth)$$
$$Believes(Solomon, Round(Earth))$$
$$\models Round(Earth)$$

If $Round(Earth)$ is an atomic wff, it's not a term, and only terms may be arguments of functions and predicates.

Even if it could:

$[\![Round(Earth)]\!]$ = True if $[\![Earth]\!] \in [\![Round]\!]$, else False.

So $[\![Believes(Solomon, Round(Earth))]\!]$ = True
iff $\langle [\![Solomon]\!], \textit{True-or-False}\rangle \in [\![Believes]\!]$

Reifying Propositions and the *Holds* Predicate

So how can we represent in FOL
"*Everything that Solomon believes is true*"?

- Reify (some) propositions.
  Make them objects in the domain.
  Represent them using individual constants or functional terms.
- Use $Holds(P)$ to mean
  "*P holds (is true) in the given situation*".
- Examples:

  $\forall p(Believes(Solomon, p) \Rightarrow Holds(p))$
  $Believes(Solomon, Round(Earth)) \Rightarrow Holds(Round(Earth))$

Semantics of the *Holds* Predicate

$\forall p(Believes(Solomon, p) \Rightarrow Holds(p)) \wedge Believes(Solomon, Round(Earth))$
$\Rightarrow Holds(Round(Earth))$

Type 0 individuals and terms:
$[Solomon] = [\![Solomon]\!]$ = A person named Solomon
$[Earth] = [\![Earth]\!]$ = The planet Earth
$[Round(Earth)] = [\![Round(Earth)]\!]$ = The proposition that the Earth is round

Type 1 objects and symbols:
p: A variable ranging over type 0 propositions
$[\![Round]\!]$ = A function from type 0 physical objects to type 0 propositions.
$[\![Holds]\!]$ = A set of type 0 propositions.
$[\![Believes]\!]$ = A set of pairs, type 0 People $\times$ type 0 propositions

Type 1 atomic formulas:
$[Holds(x)]$ = The type 1 proposition that $[x]$ is True.
$[\![Holds(x)]\!]$ = True if $[\![x]\!] \in [\![Holds]\!]$; False otherwise
$[Believes(x, y)]$ = The type 1 proposition that $[x]$ believes $[y]$
$[\![Believes(x, y)]\!]$ = True if $\langle\ [\![x]\!], [\![y]\!]\ \rangle \in [\![Believes]\!]$; False otherwise