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

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

$$\forall x(Driver(x) \Rightarrow \neg Passenger(x))$$
$$\forall x(Passenger(x) \Rightarrow \neg Driver(x))$$
$$\forall x, y(Drives(x, y) \Leftrightarrow (Driver(x) \land Passenger(y)))$$

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.

Skolem Constants:

- 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: $f, g_2, motherOf, familyOf$.

Skolem Function Symbols:

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

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 5

Each Function Symbol, Skolem 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 6

In any specific predicate logic language

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

must be disjoint.

Terms

- Every individual constant, every Arbitrary Individual,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.)
  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
  $(P \wedge Q)$ $(P \vee Q)$
  $(P \Rightarrow Q)$ $(P \Leftrightarrow Q)$
- If P is a wff and x is a variable, then $\forall x(P)$ and $\exists x(P)$ are wffs.

Open, Closed, and Ground

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.

4.2.2 Substitutions

Notice, terms may now include functional terms.

E.g.:

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

$$\{x/y, f(a, y)/z\} \circ \{u/y, g(v)/w\} = \{x/y, f(a, u)/z, g(v)/w\}$$

4.2.3 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 Constants:
$[\![a]\!]$ is some particular individual in $\mathcal{I}$.

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

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

Semantics of Functional Terms

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

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

Skolem Function Symbols:

$[\![f_s^n]\!]$ is a representative of some function in $\mathcal{D}$,

$\underbrace{\mathcal{I} \times \cdots \times \mathcal{I}}_{n \text{ times}} \to \mathcal{I}$, but it's unspecified which one.

Functional Terms:

If f^n is some function symbol or Skolem 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 particular subset of $\mathcal{I}$.
- $[\![P^n]\!]$ is some particular subset of the relation

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

Semantics of 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

- $[\![\neg P]\!]$ is True if $[\![P]\!]$ is False. Otherwise, it is False.
- $[\![P \wedge Q]\!]$ is True if $[\![P]\!]$ is True and $[\![Q]\!]$ is True. Otherwise, it is False.
- $[\![P \vee Q]\!]$ is False if $[\![P]\!]$ is False and $[\![Q]\!]$ is False. Otherwise, it is True.
- $[\![P \Rightarrow Q]\!]$ is False if $[\![P]\!]$ is True and $[\![Q]\!]$ is False. Otherwise, it is True.

Semantics of WFFs, Part 2

- $[\![P \Leftrightarrow Q]\!]$ is True if $[\![P]\!]$ and $[\![Q]\!]$ are both True, or both False. Otherwise, it is False.

- $[\![\forall x P]\!]$ 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 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 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 CarPool World Situation

$[\![Tom]\!]$ = the individual named Tom.
$[\![Betty]\!]$ = the individual named Betty.
$[\![Driver]\!] = \{motherOf(Betty), motherOf(Tom)\}$.
$[\![Passenger]\!] = \{Betty, Tom\}$.
$[\![Drives]\!] = \{\langle motherOf(Betty), Betty\rangle, \langle motherOf(Tom), Tom\rangle\}$.

4.2.4 Hilbert-Style Axioms 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 $\neg \forall x \neg \mathcal{A}$.

Fitch-Style Natural Deduction

As for Elementary Predicate Logic

$$
\begin{array}{r|ll}
i. & \underline{\;a\;\;\;\;} & ArbI \\
 & \vdots & \\
j. & A & \\
\multicolumn{1}{r}{j+1.} & \multicolumn{1}{l}{\forall x A\{x/a\}} & \forall I, i\text{–}j
\end{array}
\qquad
\begin{array}{r|ll}
i. & \forall x A & \\
i+1. & A\{t/x\} & \forall E, i
\end{array}
$$

$$
\begin{array}{r|ll}
i. & A(t) & \\
i+1. & \exists x A(x) & \exists I, i
\end{array}
\qquad
\begin{array}{r|ll}
i. & \exists x A & \\
i+1. & A\{s/x\} & \exists E, i
\end{array}
$$

Where a is an arbitrary individual not otherwise used in the proof, and s is a Skolem constant not otherwise used in the proof,

CarPool Situation Derivation

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

Model Checking in Full FOL

n Individual Constants.

At least one function yields ∞ terms.
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.

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, Skolem Constants, Function Symbols, Skolem Function Symbols, Predicate Symbols, Ground Terms, and Ground Atomic Formulas as for Standard FOL.

- 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**.

4.3.3 Direct Inference
Rules of Inference
Version 1

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

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

Example Derivation

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

Motivation for a Shortcut

$$\frac{\begin{array}{cc} \{P(x), L_1(x), \ldots, L_n(x)\} & \{\neg P(y), L_{n+1}(y), \ldots, L_m(y)\} \\ \downarrow \{a/x, a/y\} & \downarrow \{a/x, a/y\} \\ \{P(a), L_1(a), \ldots, L_n(a)\} & \{\neg P(a), L_{n+1}(a), \ldots, L_m(a)\} \end{array}}{\{L_1(a), \ldots, L_n(a), L_{n+1}(a), \ldots, L_m(a)\}}$$

Most General Unifier

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

is a substitution, μ,

such that $\mathcal{A}\mu = \mathcal{B}\mu$

$=$ a most general common instance of $\mathcal{A}$ and $\mathcal{B}$

Rule of Inference
Version 2

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.

Example Derivation Revisited

1. $\{\neg Drives(x,y), Driver(x)\}$ $\quad$ $Assumption$
2. $\{\neg Driver(x), \neg Passenger(x)\}$ $\quad$ $Assumption$
3. $\{Drives(motherOf(Tom), Tom)\}$ $\quad$ $Assumption$

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

Unification

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

Some Examples:

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

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 `((,var . ,newterm))))))))
```

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 Refutation Resolution 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(80): (prove '((all x (all y ((~ (Drives x y))
                                    or (Driver x))))
                     (all x ((~ (Driver x))
                              or (~ (Passenger x))))
                     (Drives (motherOf Tom) Betty))
                    '(~ (Passenger (motherOf Tom))))

 1  ((Drives (motherOf Tom) Betty)) Assumption
 2  ((~ (Drives ?2 ?3)) (Driver ?2))  Assumption
 3  ((~ (Driver ?5)) (~ (Passenger ?5)))  Assumption
 4  ((Passenger (motherOf Tom)))  From Query
 5  ((~ (Driver (motherOf Tom)))) R  4  3 ((?5 motherOf Tom))
 6  ((~ (Drives (motherOf Tom) ?6)))  R  5  2 ((?2 motherOf Tom))
 7  nil              R  6  1 ((?6 . Betty))
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
```

Predicate Refutation Resolution is Incomplete

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

$\vdots$

Rules of Inference
Version 3

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.)

Refutation with Resolution & Factoring is Complete

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.	$\{\}$	$R, 3, 4, \{w/z\}$

Factoring (Condensing) by `snark`

```
snark-user(95): (initialize)
; Running on #x80fdafc5 at 2004-02-16T15:34:51
nil
snark-user(96): (assert '(or (P ?u) (P ?v)))
nil
snark-user(97): (prove '(and (P ?x) (P ?y)))

(Row 1 (P ?x) (condense assertion))
(Row 2 false (rewrite ~conclusion 1))

(Refutation
(Row 1 (P ?x) (condense assertion))
(Row 2 false (rewrite ~conclusion 1))
)
:proof-found
```

Efficiency Rules

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

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.

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.

These rules delete unhelpful clauses.

Subsumption may be required to cut infinite loops.

Subsumption Cutting a Loop

```
prover(5): (prove '((all x
                          (all y
                               (all z
                                    (((ancestor x y)
                                      & (ancestor y z))
                                     => (ancestor x z))))))
                   '(exists x (ancestor x stu)))

 1  ((~ (ancestor ?3 ?4)) (~ (ancestor ?4 ?5))
     (ancestor ?3 ?5))                            Assumption
 2  ((~ (ancestor ?7 stu)))                       From Query
```

Resolution & Subsumption Steps

```
 1  ((~ (ancestor ?3 ?4)) (~ (ancestor ?4 ?5))
     (ancestor ?3 ?5))             Assumption
 2  ((~ (ancestor ?7 stu)))        From Query
 3  ((~ (ancestor ?8 ?9))
     (~ (ancestor ?9 stu)))        R  2  1 ((?5 . stu) (?7 . ?3))
 4  ((~ (ancestor ?10 stu)) (~ (ancestor ?11 ?12))
     (~ (ancestor ?12 ?10)))       R  3  1 ((?9 . ?5) (?8 . ?3))
 5  ((~ (ancestor ?13 ?14)) (~ (ancestor ?14 ?15))
     (~ (ancestor ?15 stu)))       R  3  1 ((?5 . stu) (?9 . ?3))

Deleting  5 because it's subsumed by  4
Deleting  4 because it's subsumed by  2
Deleting  3 because it's subsumed by  2
nil
```

Strategies

Unit Preference: Resolve shorter clauses before longer clauses.

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

Many others

These are heuristics for finding $\{\}$ faster.

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, 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

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

From: $\forall x[Parent(x) \Leftrightarrow (Person(x) \land \exists y(Person(y) \land Child(y,x)))]$

To:

$$\forall x[(Parent(x) \Rightarrow (Person(x) \land \exists y(Person(y) \land Child(y,x)))) \\ \land((Person(x) \land \exists y(Person(y) \land Child(y,x))) \Rightarrow Parent(x))]$$

Translation Step 2

Eliminate occurrences of $\Rightarrow$ using

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

From:

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

To:

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

Translation Step 3

Translate to *miniscope* form using

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

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

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

From:

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

To:

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

Translation Step 4

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

From:

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

To:

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

Translation Step 5

Skolemize

From:

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

To:

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

Translation Step 6

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

From:

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

To:

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

Translation Step 7

CNF: Translate into Conjunctive Normal Form, using

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

From:

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

To:

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

Translation Step 8

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

From:

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

To:

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

Translation Step 9

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

From:

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

To:

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

Translation Step 10

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

From:

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

To:

$$\{\{\neg Parent(x_1), Person(x_1)\},$$
$$\{\neg Parent(x_2), Person(f(x_2))\},$$
$$\{\neg Parent(x_3), Child(f(x_3), x_3)\},$$
$$\{\neg Person(x_4), \neg Person(z_4), \neg Child(z_4, x_4), Parent(x_4)\}\}$$

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 `prover`

```
(prove
 '((all x ((Parent x)
          <=> ((Person x)
               & (exists y ((Person y) & (Child y x))))))
   (Person Tom) (Person Betty) (Child Tom Betty))
 '(Parent Betty))

 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((Child Tom Betty))                         Assumption
 4  ((~ (Parent ?3)) (Person ?3))               Assumption
 5  ((~ (Parent ?4)) (Person (S2 ?4)))          Assumption
 6  ((~ (Parent ?5)) (Child (S2 ?5) ?5))        Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))             Assumption
 8  ((~ (Parent Betty)))                        From Query
```

Resolution Steps

```
 1  ((Person Tom))                         Assumption
 2  ((Person Betty))                       Assumption
 3  ((Child Tom Betty))                    Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))        Assumption
 8  ((~ (Parent Betty)))                   From Query
 9  ((~ (Person Betty)) (~ (Person ?8))
     (~ (Child ?8 Betty)))                 R  8  7 ((?6 . Betty))
13  ((~ (Person Betty)) (~ (Person Tom))) R  9  3 ((?8 . Tom))
19  ((~ (Person Tom)))                     R 13  2 nil
22  nil                                    R 19  1 nil
QED
```

Connections

Modus Ponens

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

Resolution

$$\frac{\{A\}, \{\neg A, B\}}{\{B\}}$$

Modus Tollens

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

Resolution

$$\frac{\{\neg A, B\}, \{\neg B\}}{\{\neg A\}}$$

Disjunctive Syllogism

$$\frac{A \vee B, \neg A}{B}$$

Resolution

$$\frac{\{A, B\}, \{\neg A\}}{\{B\}}$$

Chaining

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

Resolution

$$\frac{\{\neg A, B\}, \{\neg B, C\}}{\{\neg A, C\}}$$

More Connections

Clause	**Rule**
$\{\neg A_1, \ldots, \neg A_n, C_1, \ldots, C_m\}$	$(A_1 \wedge \cdots \wedge A_n) \Rightarrow (C_1 \vee \cdots \vee C_m)$
Horn Clause	**Rule**
$\{\neg A_1, \ldots, \neg A_n, C\}$	$(A_1 \wedge \cdots \wedge A_n) \Rightarrow C$
	Prolog Clause
	C :- $A_1, \ldots, A_n$
Set of Support	**Back-chaining**

4.5 Asking Wh Questions

Given

$\forall x[Parent(x) \Leftrightarrow (Person(x) \land \exists y(Person(y) \land Child(y, x)))]$
$Person(Tom)$
$Person(Betty)$
$Child(Tom, Betty)$

Ask: “Who is a parent?”

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

Try to Answer Wh Question

```
(prove
 '((all x ((Parent x)
          <=> ((Person x)
               & (exists y ((Person y) & (Child y x))))))
   (Person Tom) (Person Betty) (Child Tom Betty))
 '(exists x (Parent x)))

 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((Child Tom Betty))                         Assumption
 4  ((~ (Parent ?3)) (Person ?3))               Assumption
 5  ((~ (Parent ?4)) (Person (S2 ?4)))          Assumption
 6  ((~ (Parent ?5)) (Child (S2 ?5) ?5))        Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))             Assumption
 8  ((~ (Parent ?9)))                           From Query
```

Resolution Steps

```
 1  ((Person Tom))                     Assumption
 2  ((Person Betty))                   Assumption
 3  ((Child Tom Betty))                Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))    Assumption
 8  ((~ (Parent ?9)))                  From Query
 9  ((~ (Person ?10)) (~ (Person ?11))
     (~ (Child ?11 ?10)))              R  8  7 ((?9 . ?6))
14  ((~ (Person Betty))
     (~ (Person Tom)))                 R  9  3 ((?10 . Betty)
                                                 (?11 . Tom))
16  ((~ (Person Tom)))                 R 14  2 nil
17  nil                                R 16  1 nil
QED
```

The Answer Predicate

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

and refutation resolution 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 $\{(Answer \ldots) \cdots (Answer \ldots)\}$.

Using the Answer Predicate

```
(prove
 '((all x ((Parent x)
          <=> ((Person x)
               & (exists y ((Person y) & (Child y x))))))
   (Person Tom) (Person Betty) (Child Tom Betty))
 '(exists x (Parent x)))
```

```
 1  ((Person Tom))                                   Assumption
 2  ((Person Betty))                                 Assumption
 3  ((Child Tom Betty))                              Assumption
 4  ((~ (Parent ?3)) (Person ?3))                    Assumption
 5  ((~ (Parent ?4)) (Person (S2 ?4)))               Assumption
 6  ((~ (Parent ?5)) (Child (S2 ?5) ?5))             Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))                  Assumption
 8  ((~ (Parent ?9)) (Answer (Parent ?9))) From Query
```

Resolution Steps

```
 1  ((Person Tom))                                  Assumption
 2  ((Person Betty))                                Assumption
 3  ((Child Tom Betty))                             Assumption
 7  ((~ (Person ?6)) (~ (Person ?7))
     (~ (Child ?7 ?6)) (Parent ?6))                 Assumption
 8  ((~ (Parent ?9)) (Answer (Parent ?9)))          From Query
 9  ((Answer (Parent ?10)) (~ (Person ?10))
     (~ (Person ?11)) (~ (Child ?11 ?10)))         R  8  7 ((?9 . ?6))
14  ((Answer (Parent Betty))
     (~ (Person Betty)) (~ (Person Tom)))       R  9  3 ((?10 . Betty)
                                                         (?11 . Tom))
21  ((Answer (Parent Betty)) (~ (Person Tom))) R 14  2 nil
24  ((Answer (Parent Betty)))                   R 21  1 nil
QED
```

Answer Predicate in `snark`

```
snark-user(11): (assert '(all x (iff (Parent x)
                                   (exists y (and (Person y)
                                                  (Child y x))))))
nil
snark-user(12): (assert '(Person Tom))
nil
snark-user(13): (assert '(Person Betty))
nil
snark-user(14): (assert '(Child 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 (Child ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (Child Tom Betty) assertion)
(Row 7 (not (Parent ?x)) ~conclusion
   Answer (Parent ?x))
(Row 8 (or (not (Person ?x)) (not (Child ?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 (Child ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (Child Tom Betty) assertion)
(Row 10 (not (Parent ?x)) ~conclusion
   Answer (Parent ?x))
(Row 11 (or (not (Person ?x)) (not (Child ?x ?y)))
   (resolve 10 3)
   Answer (Parent ?y))
(Row 12 false (rewrite (resolve 11 6) 4)
   Answer (Parent Betty))
)
(Parent Betty)
```

Disjunctive Answers

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         ((Red b) or (Green b)))
       '(exists x
         (exists y ((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  ((~ (Red ?2)) (~ (Green ?3)) (~ (On ?2 ?3))
     (Answer ((Red ?2) & ((Green ?3) & (On ?2 ?3))))) From Query
```

Resolution Steps

```
 9  ((~ (Green ?4)) (~ (On a ?4))
     (Answer ((Red a) & ((Green ?4) & (On a ?4)))))
                                                    R  6  3 ((?2 . a))

10  ((~ (Red ?5)) (~ (On ?5 c))
     (Answer ((Red ?5) & ((Green c) & (On ?5 c)))))
                                                    R  6  4 ((?3 . c))

11  ((Green b) (~ (Green ?6)) (~ (On b ?6))
     (Answer ((Red b) & ((Green ?6) & (On b ?6)))))
                                                    R  6  5 ((?2 . b))

13  ((~ (Red b))
     (Answer ((Red b) & ((Green c) & (On b c)))))
                                                    R 10  2 ((?5 . b))
```

Resolution Finished

```
16  ((Green b)
     (Answer ((Red b) & ((Green c) & (On b c))))) R 13  5 nil

20  ((Answer ((Red b) & ((Green c) & (On b c))))
     (~ (On a b))
     (Answer ((Red a) & ((Green b) & (On a b)))))
                                          R  9 16 ((?4 . b))

22  ((Answer ((Red b) & ((Green c) & (On b c))))
     (Answer ((Red a) & ((Green b) & (On a b))))) R 20  1 nil
QED
```

Multiple Clauses From Query

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         ((Red b) or (Green b)))
       '(exists x ((Red x) or (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 ((~ (Red ?1))
   (Answer ((Red ?1) or (Green ?1))))        From Query
 7 ((~ (Green ?2))
   (Answer ((Red ?2) or (Green ?2))))        From Query
 8 ((Answer ((Red a) or (Green a))))         R  6  3 ((?1 . A))
QED
```

Resolution Produces Only 1 Answer

```
snark-user(20): (initialize)
; Running on #x80fdafc5 at 2004-02-23T09:27:06
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)
```

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. $\forall r[Transitive(r) \Leftrightarrow \forall xyx[r(x,y) \wedge r(y,z) \Rightarrow r(x,z)]]$
is not a first-order sentence.

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 terms denoting individuals; Assign type $i+1$ to any function or predicate symbol that takes any argument of type i. I.e., no predicate or function of type i may take an argument of type i or higher. First-order logic is limited to symbols of type 1. n^{th}-order logic is limited to symbols of type n. Ω-ordered logic has no limit, but must still follow the rules.

5 Summary

- **Logic** is the study of correct reasoning.
- There are many systems of logic (logics).

Parts of a Logic

- Syntax
- Semantics
  - Intensional: Formalizing a Domain
  - Extensional: Specifying a Situation
- Proof Theory

Soundness is the essence of "correct reasoning."

Relations Among Inference Methods

Syntax	**Derivation**		**Theoremhood**
	$A_1, \ldots, A_n \vdash P$	$\Leftrightarrow$	$\vdash A_1 \wedge \ldots \wedge A_n P$
	$\Downarrow\Uparrow$		$\Downarrow\Uparrow$
	$A_1, \ldots, A_n \models P$	$\Leftrightarrow$	$\models A_1 \wedge \ldots \wedge A_n P$
Semantics	**Logical Implication**		**Validity**

Inference Methods

- Direct, syntactic methods apply rules of inference to axioms (logical or non-logical).

- Refutation, semantic methods attempt to

  - fail at attempt to find a situation that satisfies the negation of the query.

  - or show the negation of the query is contradictory. (There is no situation that satisfies it.)

- Resolution (plus Factoring) is a complete refutation semantic inference method.

Unification

- Unification is a least-commitment method of choosing a substitution for Universal Instantiation ($\forall E$).

- Unification is correct pattern matching when both structures have variables.

- Unification is generally needed in backward chaining systems.

Effect of Functions

- Inclusion of functions makes inference hard (semi-decidable).

- Existential quantifiers lead to functions (Skolem functions).

AI-Logic Connections

AI	Logic
rules or domain rules	non-atomic assumptions or non-logical axioms
inference engine procedures	rules of inference
knowledge base	proof

Relevance of Logic

Any system that consists of

- a collection of symbol structures
- a set of procedures for adding new structures to that collection based on what's already in there

is a logic.

Do the symbol structures have a consistent semantics?

Are the procedures sound relative to that semantics?