CIS 730 Artificial Intelligence
CIS 530 Introduction to Artificial Intelligence
Fall 2013

Homework 4 of 8: Machine Problem
Knowledge Representation and Reasoning, Part I:
First-Order Logic to CNF Conversion and Ontology Development

Assigned: Sat 05 Oct 2013
Due: Fri 18 Oct 2013 (before midnight)

The purpose of this assignment is to exercise your basic understanding of first-order logic and ontology engineering (design and inference) through a simple implementation. You will also start to write a program to partially parse first order predicate calculus (FOPC), *aka* first-order logic (FOL) sentences, into CNF, i.e., clausal form. This will be completed and applied in Machine Problem 5.

This homework assignment is worth a total of 100% (20 points).
Each problem is worth 25% (4 points) for CIS 730 students and 33% (7 points) points for CIS 530 students.

Conjunctive Normal Form (CNF) converter

Refer to the following specification for Problems 1-3.

```
The hash mark denotes comments.
# Your program should ignore all input on a line containing the '#'
# symbol.
#
# The first non-comment line contains a single integer
# denoting the number of sentences.
3
# The second non-comment line starts the sentences.
# Syntax:
#     \A denotes UNIVERSAL QUANTIFICATION (\forall)
#     \E denotes EXISTENTIAL QUANTIFICATION (\exists)
#     => denotes IMPLICATION (\rightarrow)
#     & denotes CONJUNCTION, i.e., AND
#     | denotes DISJUNCTION, i.e., OR (\vee)
#     ! denotes NEGATION, i.e., NOT (\not)
#     = denotes EQUALITY (NOTE: this is OPTIONAL in the regular MP)
#     . separates each quantified variable from its scoped expression
# Conventions:
#     - All variables are lowercase
#     - All constants are alphanumeric names in ALL CAPS
#     - All predicates contain alphanumeric characters or _, begin with
#       a capital letter, and enclose their arguments in parentheses
#     - All functions contain alphanumeric characters or _, begin with
#       a lowercase letter, and enclose their arguments in parentheses
#     - Use C/C++ precedence for & and |
\A x . \A y . P(x, y) => \E z . Q(x, z) & !R(y, z)
Husband_Of (JOE1, SUSAN)
Longer (left_leg_of(RICHARD), left_leg_of(JOHN)) & (Foo() | bar = baz)
#
# Correct answers:
# 1. {!P(x_1, y_1), Q(x_1, sf1(x_1, y_1))},
```

```
#     {!P(x_2, y_2), !R(x_2, sf1(x_2, y_2))}
# 2.  {Husband_Of (JOE1, SUSAN)}
# 3.  {{Longer (left_leg_of(RICHARD), left_leg_of(JOHN))},
#      {Foo(), bar = baz}}
```

1. **(530/730) Scanning and parsing FOL expressions.** Write a Java or C++ program to scan and parse the above expressions into an internal *expression tree* format. You may use any of the following parser and scanner generators:

   a) Java string tokenizer
   http://java.sun.com/j2se/1.4.2/docs/api/java/util/StringTokenizer.html
   b) C Standard Library (`string.h`)
   http://www.cs.cf.ac.uk/Dave/C/node19.html
   c) C++ Standard Template Library (STL) string library
   http://www.processdoc.com/doc/cppstl/string.html
   d) `lex/flex, ml-lex`
   http://cs.wwc.edu/~aabyan/464/Book/LexFlex.html
   See also: http://dinosaur.compilertools.net
   http://www.smlnj.org/doc/ML-Lex/
   e) `yacc/bison, ml-yacc`
   http://cs.wwc.edu/~aabyan/464/Book/YaccBison.html
   See also: http://dinosaur.compilertools.net
   http://smlnj.cs.uchicago.edu/doc/ML-Yacc/
   f) *Antlr*.
   http://www.antlr.org
   g) *PyParsing* or one of the other Python-based parsers listed at the Python wiki:
   http://pyparsing.wikispaces.com
   https://wiki.python.org/moin/LanguageParsing

   As output, print the expression tree in a *prefix traversal*. For example, P | Q should be printed out as Or (P, Q). Turn in `mp4_3` (prefix any Lex, Yacc, or Antlr spec thusly) and include instructions in your README.txt file for generating your actual scanner/parser source, compiling it, and running it on test input.

2. **(530/730) Implications, deMorgan's Theorem, and standardization of variable names.** Extend your parser code by adding code to perform the "INS" part of the "INSEUDOR" procedure for conversion to clausal form. Print the resulting intermediate version of expressions given as standard input. Turn in `mp4_4`, a revised version of your scanner/parser code or spec.

3. **(730) Skolemization.** Perform full Skolemization as explained in Russell and Norvig 2^e, and in the examples given in class. Turn in `mp4_5`, a revised version of your scanner/parser code or spec.

4. **(530/730 only) Ontologies.**

   Reference

   Protégé-OWL tutorial: http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/

   The exercises and page numbers in this problem refer to v1.2 of the tutorial:
   http://bit.ly/1pZGBF

   a) Download Protégé 4 from http://protege.stanford.edu and follow the instructions through Chapter 4 to create the Pizza ontology. Turn in OWL files with your solutions

to Exercises 14 (p. 35), 16-17 (p. 42 – 43), 18 (p. 45), 19-20 (p. 45 – 46), and 23 (p. 48). Also, turn in JPEG screenshots of your ontology definitions.

b) Complete exercises 24 (p. 51), 25 (p. 52), 26 (p. 53), 27 (p. 54), and 28-30 (p, 55 – 58).

5. **(730 only) Prolog.** Adapted from p. 37-41, *Logic, Programming and Prolog* by Ulf Nilsson and Jan Małuszyński. Consider the axioms:

Rules

i. *Every powerful Jedi taught by a good master follows the light side of the Force.*
ii. *Luke is a powerful Jedi.*
iii. *Yoda is a good master.*
iv. *Luke was taught by Yoda.*

Use Prolog to prove that Luke follows the light side of the Force.
Turn in your source (`mp4_1.pl`) and a problem solving trace (`mp4_1-out.txt`).

Extra credit (20%): Write a simple graph generator for MP2 in your favorite programming language. Turn in the source code and some sample graphs of type "good" (simple, small, with consistent heuristics, and without boundary conditions), "bad" (pathologically bad, with inadmissible or inconsistent heuristics), and "ugly" (large and/or dense, with $n \geq 25$, preferably $n = 50$ or $n = 100$).

Class Participation (required)

After going over your **Read And Explain Pairs** exercise with your assigned partner, post a short paragraph summarizing *unification* and a second containing any questions you may have on search to the class mailing list (**CIS530-L@listserv.ksu.edu** or **CIS730-L@listserv.ksu.edu**). That is, ask questions about any knowledge representation topic for which your understanding is unclear. This includes propositional logic, first-order logic, description logic, or theorem proving (forward and backward chaining, resolution strategies)

Midterm Exam Review

A review guide will be posted next Friday (11 Oct 2013) with examples not in PS3.

Term Project Interim Reports

Submit a short interim report as specified in class the week of Mon 14 Oct 2013 and attend office hours.

Coming Up Next

Problem Set 5 (due Mon 28 Oct 2013) – Knowledge Representation and Reasoning, Part II: Rule-Based Expert Systems, Analogical Reasoning

Machine Problem 6 (due Wed 06 Nov 2013) – Knowledge Representation and Reasoning, Part III: More Clausal Form, Rule-Based Expert Systems, and Intro to Probabilistic Reasoning (Hugin)

Problem Set 7 (due Fri 15 Nov 2013): Reasoning and Learning, Part I: Probabilistic Reasoning (Inference and Causality), Version Spaces, and Decision Trees; The Waikato Environment for Knowledge Analysis (WEKA), Artificial Neural Networks (ANNs)

Machine Problem 8 (due Fri 21 Nov 2013) – Reasoning and Learning Part II: Genetic Algorithms (GAs), Natural Language Processing (NLP), and Vision