COEN 166/266 - Spring 2015

Homework #6
Theorem Prover

Description

For this assignment, you will create an automated theorem prover. When provided with a series of facts, your program should be able to determine whether an additional fact can be validly inferred. As usual, there are several flavors of the assignment depending on how in-depth you would like to get.

Basic Assignment

Implement a statement resolution algorithm. It should accept statements in an abbreviated form of conjunctive normal form (CNF), where each argument is a list of elements that are implicitly or'd together (a disjunction of variables). The function should return one of the following responses:

1. If the two statements can resolve, it should return the resolution. For example:

```
guile> (resolve '(a (NOT b) c) '(d (NOT f) b))
(a c d (NOT f))
guile> (resolve '(a b) '((NOT b)))
(a)
```

2. If the two statements cannot be resolved, it should return False. For example:

```
guile> (resolve '(a b) '(c d))
#f
guile> (resolve '(a b) '((NOT a) (NOT b)))
#f
```

3. If the two statements resolve to a contradiction, it should return Contradiction. For example:

```
guile> (resolve '(a) '((NOT a)))
CONTRADICTION
```

Intermediate Assignment

Implement a propositional logic solver that accepts statements in CNF and then performs resolution to answer propositional logic questions. For this assignment, your solver should implement a tell and ask interface on top of the resolver from the basic assignment, using the same syntax for disjunctive statements. The ask function should return #t for a statement that can be inferred, or UNKNOWN otherwise. For example:

```
guile> (tell '((NOT a) b))
OK
guile> (tell '((NOT b) c))
OK
guile> (tell '(a))
OK
guile> (ask '(a))
#t
guile> (ask '(c))
#t
guile> (ask '(d))
UNKNOWN
```

Advanced Assignment

Implement a propositional logic solver that accepts arbitrary propositional statements, converts them into CNF and then performs resolution to answer propositional logic questions. For this assignment, the operators need only support binary and unary operations, as indicated below (i.e., you don't have to support `(OR a b c)`, but you do have to support `(OR (OR a b) c)` and `(OR a (OR b c))`).

The following operators must be supported:

1. (Unary) negation: `(NOT a)`
2. (Binary) disjunction: `(OR a b)`
3. (Binary) conjunction: `(AND a b)`
4. (Binary) implication: `(IMPLIES a b)`
5. (Binary) biconditional: `(BICONDITIONAL a b)`

Your prover should return `#t` if resolution with the negation of the premise results in a contradiction or `UNKNOWN` otherwise. Here's an example of the expected behavior:

```
guile> (tell ‘(IMPLIES hasDog hasMess))
OK
guile> (tell ‘(IMPLIES hasMess (NOT clean)))
OK
guile> (tell ‘hasDog)
OK
guile> (ask ‘hasDog)
#t
guile> (ask ‘clean)
UNKNOWN
guile> (ask ‘(NOT clean))
#t
guile> (ask ‘(IMPLIES hasDog (NOT clean)))
#t
guile> (ask ‘hasCat)
UNKNOWN
guile> (tell ‘(IMPLIES hasMess hasDog))
OK
guile> (ask ‘(BICONDITIONAL hasMess hasDog))
#t
```

Extra Credit

Take this assignment to the next level by adding some level of support for first-order logic, or a natural language interface.

Grading

The basic assignment will have a **maximum** grade of approximately 80%, an intermediate assignment a maximum grade of approximately 100%, and an advanced assignment approximately 120%.

Turn in as much as you have completed. Partial credit will be given, but make sure you complete the easier assignments before attempting the more difficult ones.

Submission

Submit your assignment to Camino.

Due Date

Saturday, May 30th at 8am